TREBALL FINAL DE GRAU

TÍTOL DEL TFG: ServiceNow Data Extraction System

TITULACIÓ: Grau en Enginyeria Telemàtica

AUTOR: Daniel Dalfó Ferrer

DIRECTOR: Jean De Leon

SUPERVISOR: Roc Meseguer

DATA: Setembre del 2020

Títol: ServiceNow Data Extraction System

Autor: Daniel Dalfó Ferrer

Director: Jean De Leon

Data: Setembre del 2020

Resum

Aquest document explica tot el procés per al desenvolupament del ServiceNow Data Extraction System, un sistema que recol·lecta dades internes de l' empresa on vaig fer les pràctiques de la carrera, les processa i finalment les presenta de forma visual i comprensible. El ServiceNow Data Extraction System està compost per diferents blocs encadenats en serie l'un darrere l'altre, en aquest document parlarem de cada un d'ells.

El ServiceNow Data Extraction System ajuda a persones sense coneixements tècnics en el camp de les dades a extreure informació valuosa a partir de col·leccions massives de dades. Aquesta informació permetrà identificar mal funcionaments dins els processos de l'empresa, detectar quins són els punts fluixos i quins són més forts i generar idees per a noves oportunitats.

El document es divideix en 5 capítols principals. Al principi s'explica la teoria i alguns conceptes relacionats amb el món de les dades necessaris per entendre les parts següents, com poden ser el "Business Intelligence" i el "Data Science". A continuació s'explica com està dissenyat cada bloc del sistema, després veurem com s'ha implementat tècnicament cada un d'aquests blocs, també s'ensenya un exemple d'ús i finalment s'extreuen conclusions.

El ServiceNow Data Extraction System a dia d'avui es segueix utilitzant dins l'empresa per analitzar el flux de treball, el funcionament dels equips i detectar problemes. El disseny explicat en aquest document és una primera versió a la qual es segueix donant suport i desenvolupant.

Title: ServiceNow Data Extraction System

Author: Daniel Dalfó Ferrer

Director: Jean De Leon

Date: September 2020

Overview

This document explains the whole process for the development of the ServiceNow Data Extraction System, a system that collects internal data from the company where I did my internship, processes it and finally presents it in a visual and understandable way. The ServiceNow Data Extraction System is made up of different blocks chained one after the other, in this document we will talk about each of them.

The ServiceNow Data Extraction System helps people with no technical knowledge in the field of data to extract valuable information from massive data collections. This information will help to identify malfunctions within the company's processes, detect which are the weak points and which are the strongest, and generate ideas for new opportunities.

The document is divided into 5 main chapters. At the beginning, the theory and some concepts related to the world of data that are required to understand the following parts, such as "Business Intelligence" and "Data Science". The following explains how each block of the system are designed, then we will see how each of these blocks has been technically implemented, an example of use is also taught, and finally conclusions are drawn.

The ServiceNow Data Extraction System is still used today in the company to analyze workflows, team operations and detect problems. The design explained in this document is a first version that is still supported and being developed.

INDEX

Introduction

Some people say that data is the new gold of this era. Nowadays analyzing data is becoming a priority for the companies, and all this interest regarding data analysis is giving rise to new increasingly popular concepts such as Business Intelligence.

Business Intelligence comprises a series of methods and technologies that companies use to have a better vision of how the business is going, how it has evolved historically and even to be able to foresee how it will go in the future.

For the last year I have been working in a company as an intern helping the Business Intelligence team, there I learned about data analysis and started growing my interest in the subject. As my interest was growing I decided to do my final project in the company and develop an internal data analysis system.

The objectives of this project are to pull data from an internal tasks managing tool used in the company, then organize, adapt, and synthesize this data to later display it in a very visual and intuitive way, so people who don't have knowledge about data science can easily understand it.

We call it ServiceNow Data Extraction System, I will call it SNDES sometimes in this document and it is a chain of tools, previously used in the company, working together to fulfill the objectives previously mentioned.

Analyzing this data we will be able to identify dysfunctions, needs, and new opportunities, to later implement an effective strategy based on insights that can provide businesses with a competitive market advantage and long-term stability.

This document is structured in five parts. First one will introduce the different blocks that compose the system and some concepts, then I will introduce the software vendors that I used. In the second chapter I will talk about the design of the system and how each part works, third chapter is about the implementation of each block in the system. The fourth chapter uses an example to show some functionalities of the ServiceNow Data Extraction System and prove that what I explained in previous chapters is working properly. The final chapter is about conclusions and thoughts that I have after all this work, how I would do the things now, what I would change, and what I learned.

Chapter 1. Definition of the system

1.1 General schema

The ServiceNow Data Extraction System, in general terms, extracts data from different sources, used by the company to organize the workflow, transform and organize this information, and finally exposes the data in a visual and interactive way, so the final user can analyze the internal workflow of the company without the need of being a data analyst.

There is no specific way to make such a system, there are many ways to do it. Below you can see a figure that describes the blocks that I used to assemble the system.

Fig. 1.1 General schema of the system

Each of these blocks are individuals, they are created by different providers and can be used for many different purposes. The reason I chose these blocks is because they are used in my company for different purposes. I will describe the blocks individually in the following chapter.

For a better understanding, I will illustrate the blocks using an easy real life example of a weather monitoring system.

1.2 Defining the blocks

1.2.1 Data source

A data source is the origin of the digital data that the system is consuming. It may be the initial location where data is born or where physical information is first digitized.
Some examples of data sources are a database, a file, live measurements from physical devices, scraped web data, or any static and streaming data services across the internet.
In the weather monitoring system example, the data source would be the sensor that collects analog signals from weather and transforms it to digital data.

1.2.2 ETL tool

Short for Extract, Transform, and Load Tool, and that is exactly what it does. These kinds of tools pull data out of one source, transforms it if it's necessary and then places it into a database.

In our example, the ETL tool could be a person that reads the results in the sensors and fulfills some forms with that information.

1.2.3 Data warehouse

The most important element in the implementation of a business intelligence system is definitely the data warehouse. This is where we will store all the information obtained from the different sources of our company, with the structure and design so that we can exploit this data for different purposes, such as the generation of reports or dashboards for further analysis.
Data is organized in conceptual warehouses known as data marts. For example companies usually organize their data warehouses in data marts like clients, sales, products, etc...

In the weather example, the data warehouse would be a folder that contains all the forms with all the data ordered by time.

1.2.4 Data Module

Data modules are containers that describe data and rules for combining and shaping data to prepare it for analysis and visualization. A data module has information from different but related subjects, it is more specific than the data warehouse.

The Data Module in our example could be a person who reads the forms, organizes them, and writes the information in documents that the person who will later interpret it will understand better.

1.2.5 Dashboards

Graphical interfaces that are created as an easy user-friendly way to analyze data in a very visual way, without the need for data science knowledge. They give us the possibility of synthesizing millions of data in graphs and also the possibility of interacting with these data using filters and other tools.

In our example, the dashboards would be the same, graphics and tables that give information about the weather.

1.3 Software vendors

I used all the technology vendors that I will explain in this section because they are the ones I had access in the company and I learned during the time since I am working there.

1.3.1 Service Now

The data source for our system is ServiceNow, a software platform that we use to manage the internal workflows, enterprise operations, and projects.

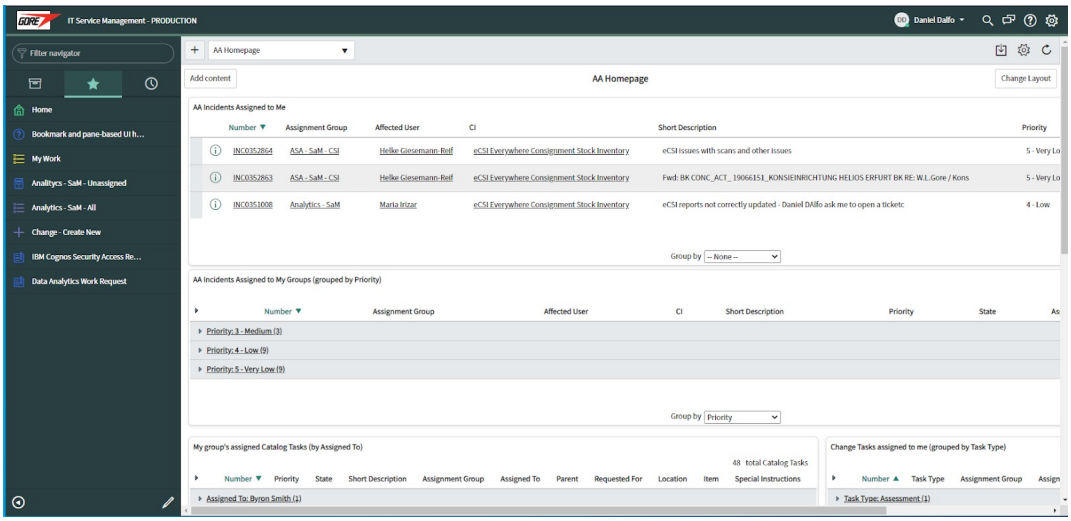


Fig. 1.2 ServiceNow homepage

It stores records for each work requested by the users to the IT team. Each of these requests will be stored as a ticket. Tickets are objects that contain information about a work that has to be done like who requested the work, the subject, who has requested the action, who will fulfill it, etc… There are few ticket types that I will explain in chapter 2.

Each associate in the IT team has their assigned tickets that have to be resolved and that is a good way to manage and organize work.

All this information is stored in a database that will be our data source, this is the data that we are pulling and using in our system.

1.3.2 Oracle Data Integrator

The data extraction process from the ServiceNow to our data warehouse is made by ODI which stands for Oracle Data Integrator, a tool produced by Oracle that offers a graphical environment to build, manage and maintain data integration processes in business intelligence systems.

Fig. 1.3 ODI logotype

To be precise ODI is an ELT tool, short for Extract, Load, and Transform tool,
It is a newer and efficient approach to populate data warehouses. In this process, data gets leveraged via a data warehouse in order to do basic transformations. That means there's no need for data staging.

In this step we pull the data from the ServiceNow database and prepare it to be stored in the data warehouse.

1.3.3 Oracle Database

We have assembled our data warehouse system using the Oracle database, the provider used by the company and one of the most popular ones. To access the database we use a very simple coding language called SQL.

Fig. 1.4 Oracle Database logotype

Here we store the data coming from the ODI organized in different tables with many columns and rows. As it is explained in more detail later, we have many tables depending on the ticket type or purpose of the data.

1.3.4 Cognos Data Module

It is a quite recent feature in Cognos 11, the data analysis system used by the company provided by IBM® Cognos Analytics. It's a web-based data acquisition system for blending and modeling data in a very visual and easy way.

Fig. 1.5 ServiceNow Data Module screenshot

Here we drop all the data that we will later analyze from ServiceNow to prepare it for the analyzing tools. The Data Module lets us create calculations, change the format of the data, add metadata, and provide deep learning tools that help the system to understand what kind of data is coming in.

I had a course on this technology as we are recently using it in the company.

1.3.5 Cognos Dashboards

This is the most visual part, here is where data adopts an user-friendly appearance and it is possible to extract conclusions.

We are using a new feature from IBM® Cognos Analytics called Dashboards that offers a series of graphical tools to show and analyze data such as tables, charts and graphs, lists, KPI's, etc…

It is an interactive and dynamic environment where the user can play with the data, to see exactly what is searching for.

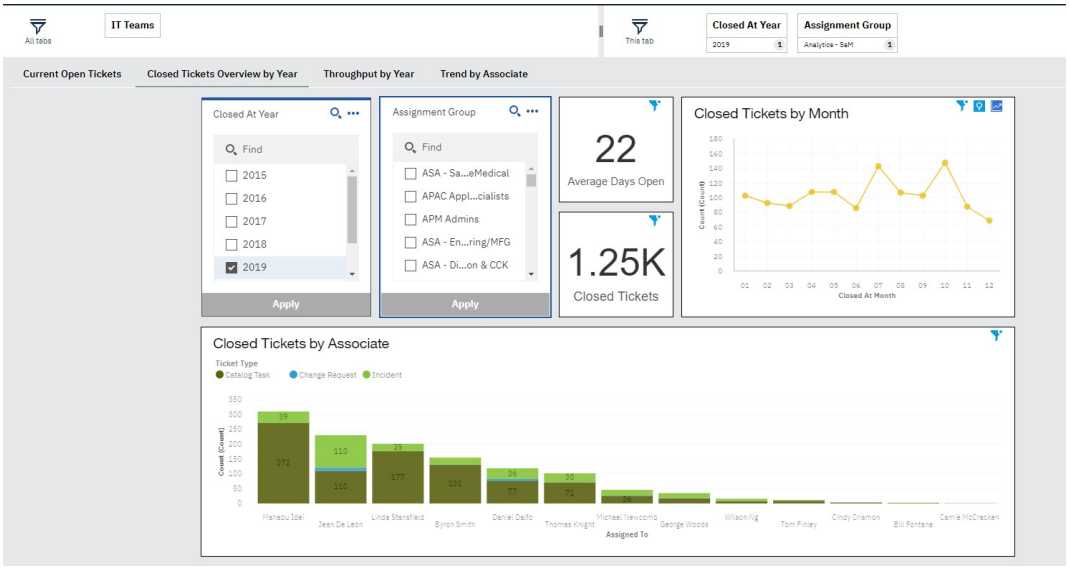


Fig. 1.6 Ticket Trend Dashboard screenshot

Chapter 2. Designing the blocks

2.1 Selecting the information

As mentioned in section 1.3.1, for each work requested, a ticket is generated.
Each associate in the company has a user account with whom they can log in the system and create a ticket. Each user working in IT belongs to a team regarding what the users are supporting.
When a ticket is created, it is assigned to a team and after that, the users working in this team will see that there is a new ticket for the team, they can read what the ticket is about and assign it to themselves or to another user or team.
Every ticket has a cycle, there are different states till the work is done, and the user closes the ticket.
Before closing a ticket, the user has to specify the work that has been done and this information will be saved in the ticket so other users with similar issues can read it.

Each user has an interface as you can see in Fig. 2.1 where all the tickets assigned to them appear. The user can interact with the tickets, check the information in the ticket, modify it and finally close it.

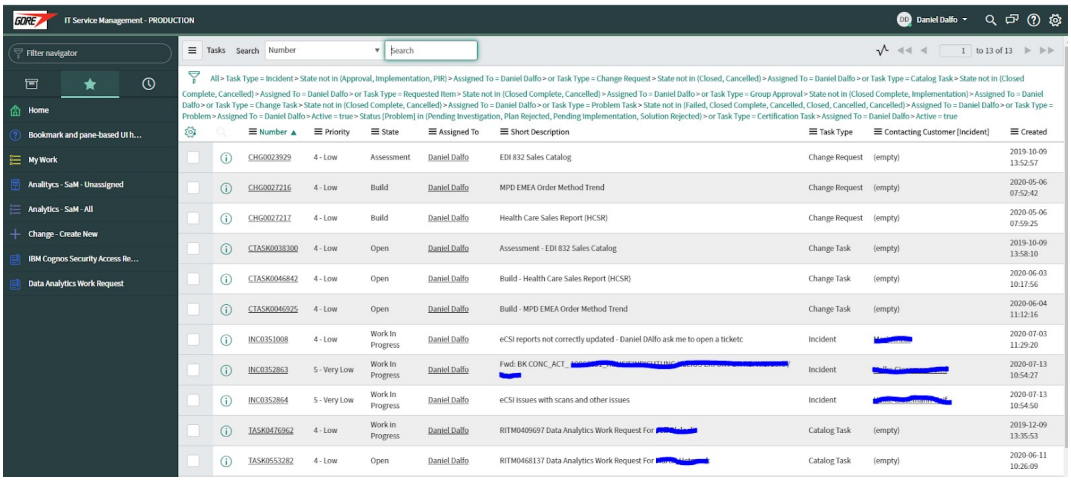


Fig. 2.1 Open tickets assigned to me

There are many ticket types, for this project I have chosen the 3 types that I use the most and I will explain them in the following sections.

2.1.1 Incident

An unplanned interruption to an IT service or reduction in the quality of an IT service. Have to be solved as fast as it's possible.

For example: A user is not receiving a report. Or the prices between two reports don't match.

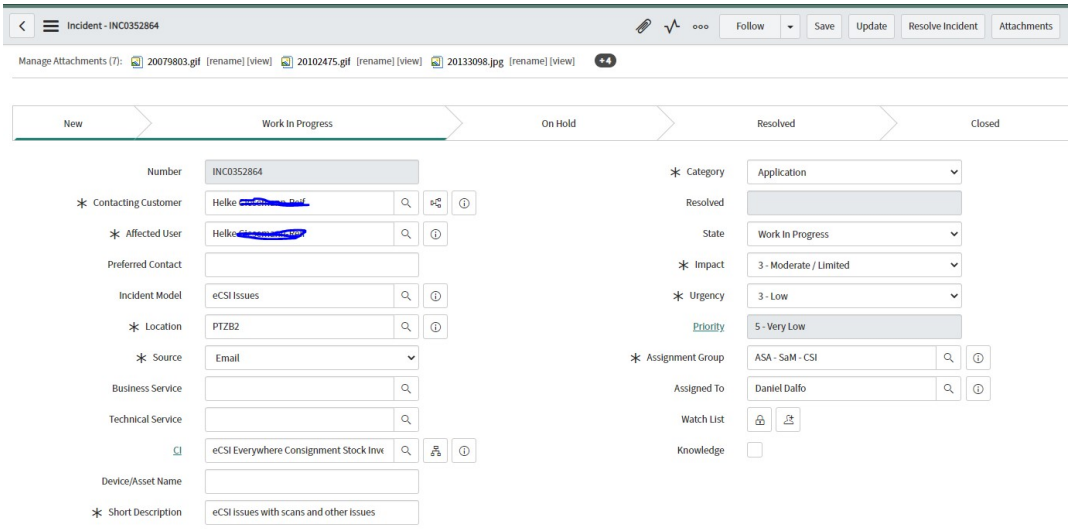


Fig. 2.2 Incident example

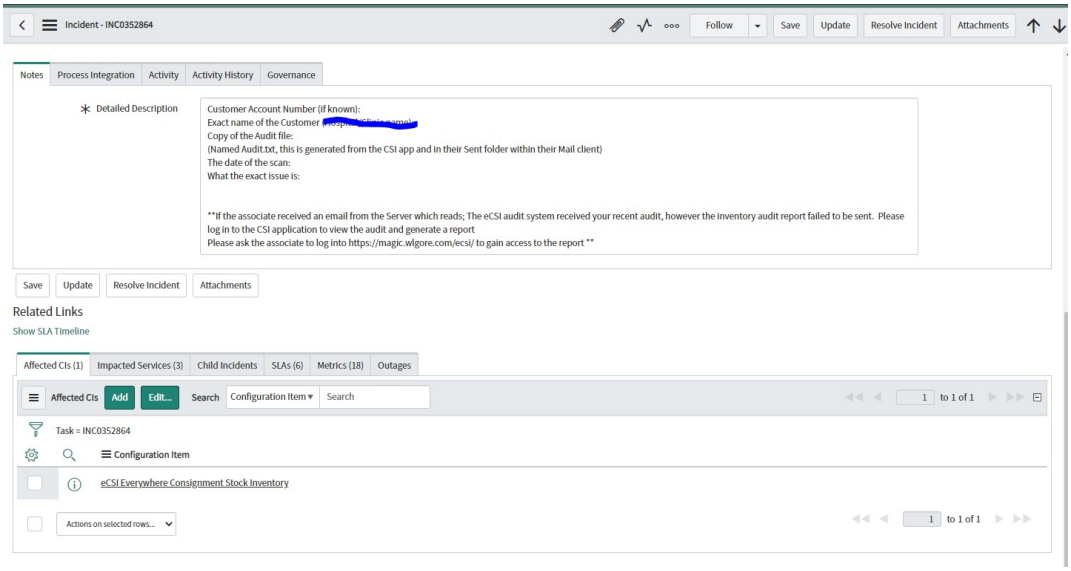


Fig. 2.3 Incident example

2.1.2 Change Request

It is categorized for drastic changes made to the IT environment.
For example: Creating a new dashboard to analyze the order trends.

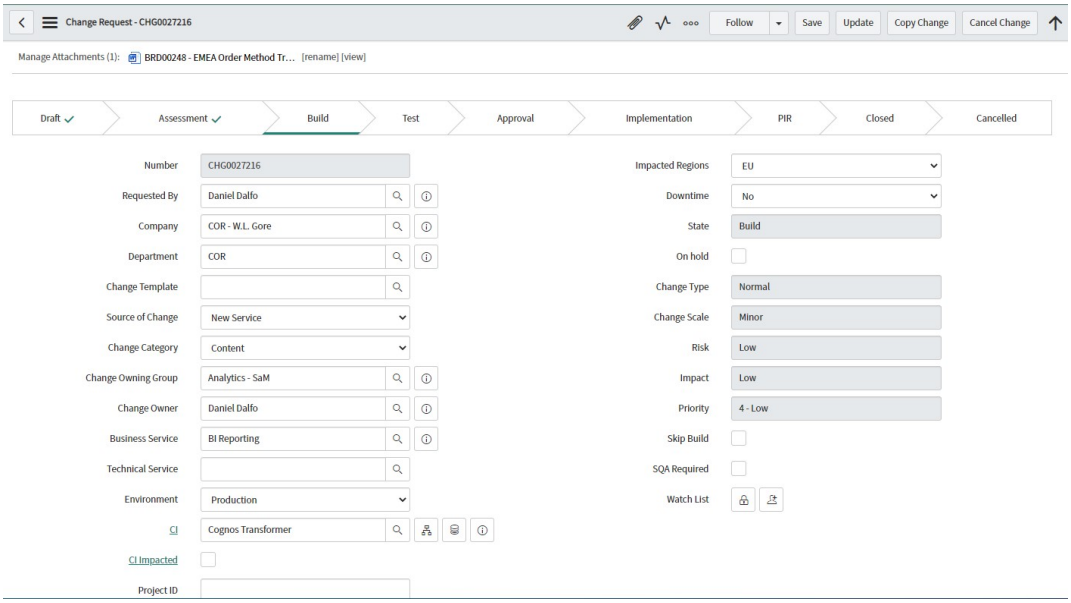


Fig. 2.4 Change Request example

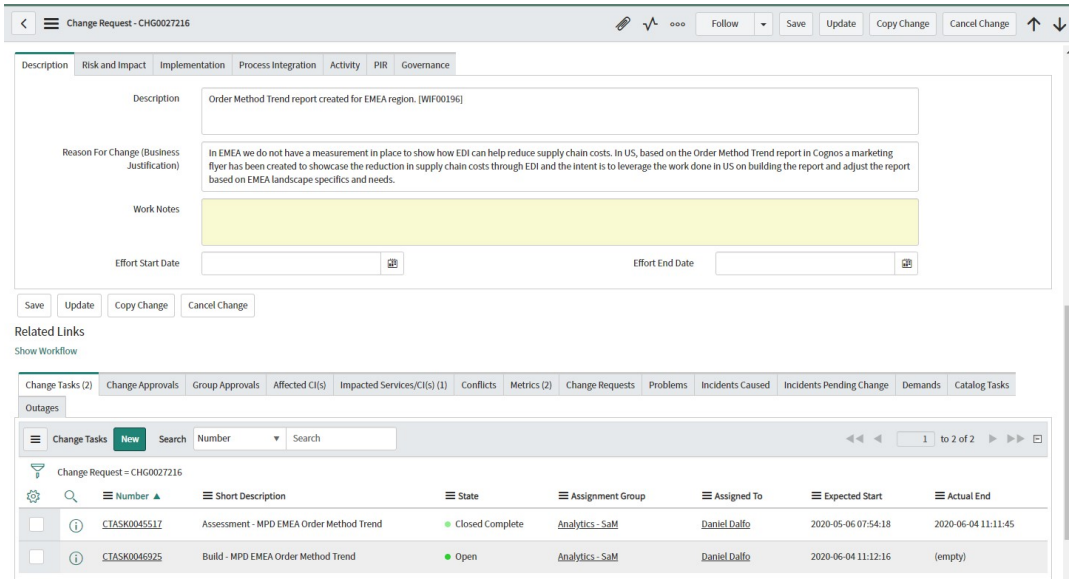


Fig. 2.5 Change Request example

2.1.3 Task

There is no disruption to a service functioning as intended, but the user is requesting something about the service be changed. Basically a task is if you want something that you don't already have, or doesn't presently exist.
For example: Add a column to this report, or grant me access to X report.

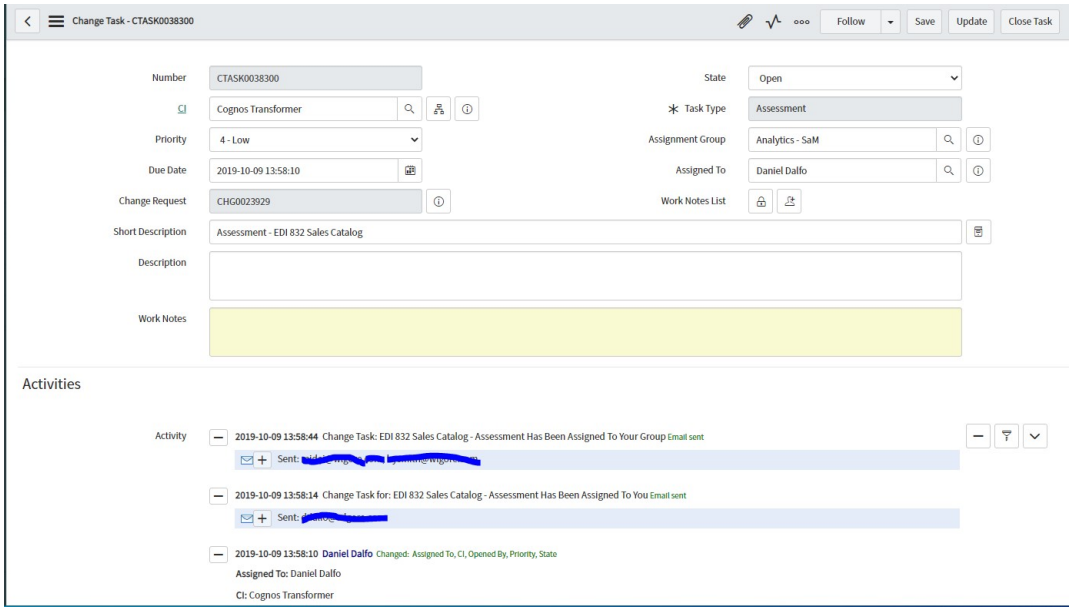


Fig. 2.6 Task example

2.2 Extraction, transform and load process design

At this point, we want to extract data from the data source (ServiceNow), adapt it to fit in our data warehouse (Oracle Database), and then save it. All this process has to be done automatically. To do so, we will use an ETL tool (Oracle Data Integrator) as mentioned in chapter 1.

2.2.1 Topology

Our topology is very simple, ODI will create a connection to the ServiceNow database and another connection to our data warehouse.

The data is created in the ServiceNow and then stored in the ServiceNow database, ODI will pull the data that we selected from the ServiceNow database, transform it, and then load it into our data warehouse.

2.2.2 Models

Once we have the connections created it is time to define the models. A model is a schema where we define the design of the tables where we will store the data coming from ServiceNow. The model defines the columns of the table, the attributes for these columns, the privileges, etc… These tables then will be stored in the data warehouse.

There is a model for each ticket type because, as I mentioned before, each ticket type stores different kinds of data and each model will have different columns.

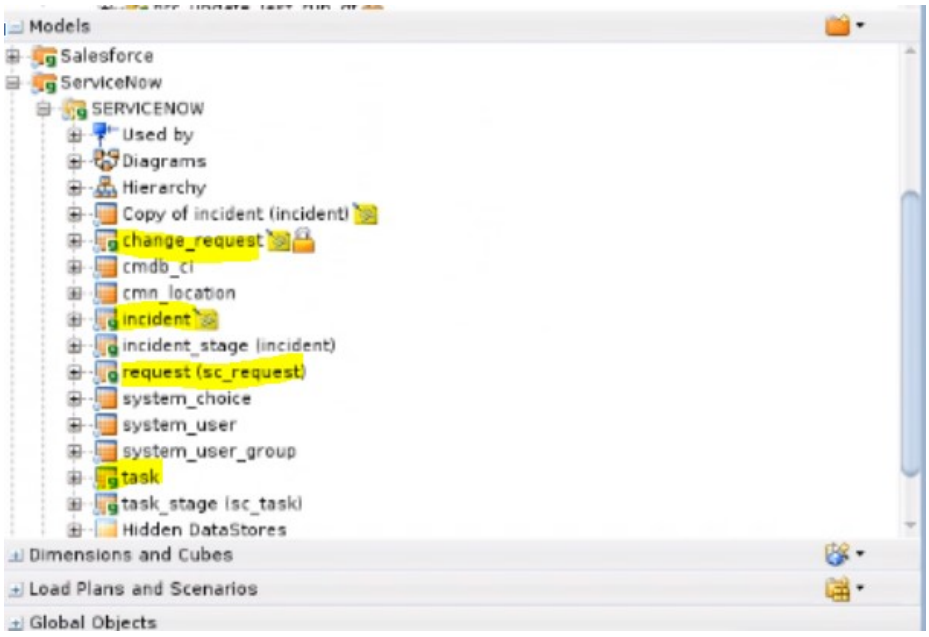


Fig. 2.7 Folder containing all the models in ODI

2.2.3 Mappings

This process matches the columns from the tables in the ServiceNow database with the columns that we defined previously in the model. A mapping process is defined for each model.

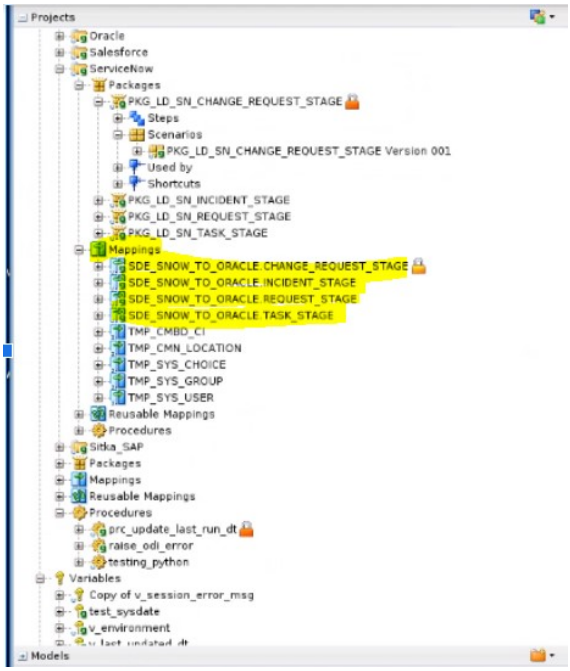


Fig. 2.8 Mappings

2.2.4 Packages

Packages in ODI are scenarios where you define the steps that the data have to follow on the data extraction process. Fig. 2.9 shows the package for the Change Request data, each object in the package contains some logic that will be applied to the data.

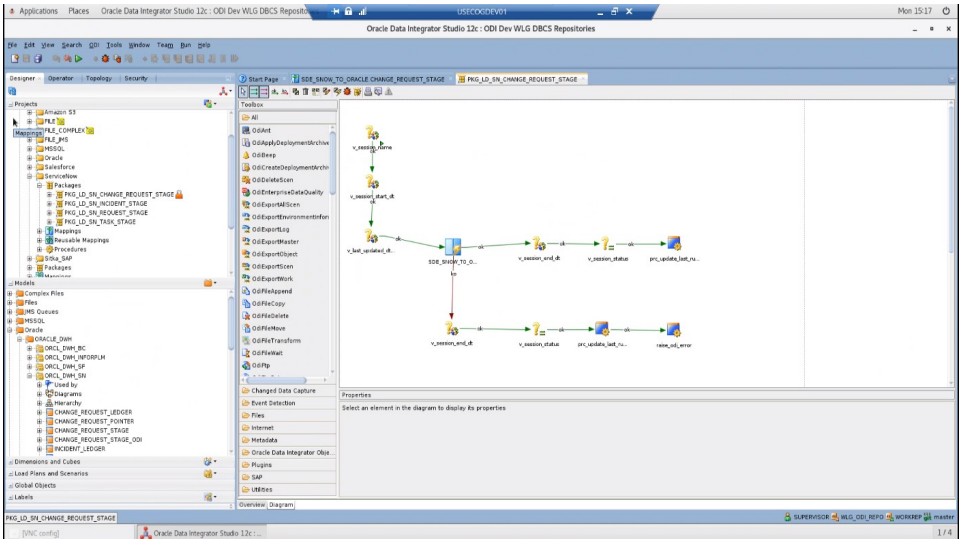


Fig. 2.9 Change Request package

Once again, we will define a package for each ticket type.

Fig. 2.10 ServiceNow packages in ODI

2.2.5 Procedure

This is the final step of the data extraction process. The procedures in ODI let you define a series of packages that have to be run at once, this is very useful to group all packages from a certain system.

We will add all the packages for every ticket type previously created inside a procedure, so when we call this procedure it will run all of them and extract all the data selected from the ServiceNow to be loaded into the data warehouse.

2.3 Designing the data warehouse

Our data warehouse is implemented on Oracle Database, the simplest way to have direct access to the data is by SQL language, and to do that I used a software called SQL Developer.
To organize our data, we are using three kind of objects:

- **Tables**: Is a collection of related data organized in columns and rows, each record is a new row.
- **Views**: Are kind of virtual tables, they also have rows and columns as a real table but are created by selecting fields from one or more tables present in the database. A View can either have all the rows of a table or specific rows based on certain conditions.
- **Materialized views**: It is very similar to a view, the difference between them and the views is that the results of a view query are not stored anywhere on disk, and the view is recreated every time the query is executed. Materialized views are actual structures stored within the database and written to disk.
- **Procedures**: Are prepared SQL codes that you can save, so the code can be reused over and over again.

It is a quite complex schema with many tables, views, materialized views, and procedures that I will explain in the next section.

2.3.1 Database schema

For each ticket type (Incident, Change Request, and Task) the following objects exist:

- A staging table
- A ledger table
- A procedure that loads data from stage to ledger tables
- A pointer table
- A backup copy for each of the previous tables
- A procedure to manage pointer tables
- A view by snapshots

There are some additional key objects in our data warehouse:

- A snapshots table
- A procedure to manage the snapshots table
- A current state combined tickets materialized view
- A monthly state combined tickets materialized view
- A monthly state combined tickets summary materialized view

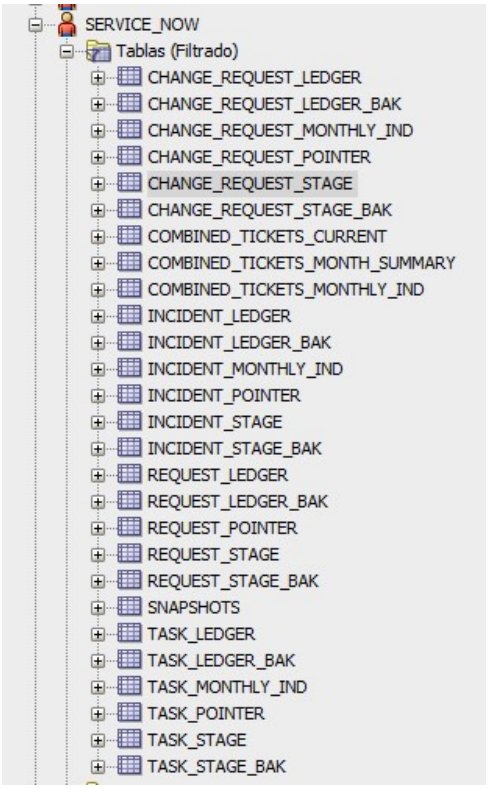


Fig. 2.11 Tables in the data warehouse

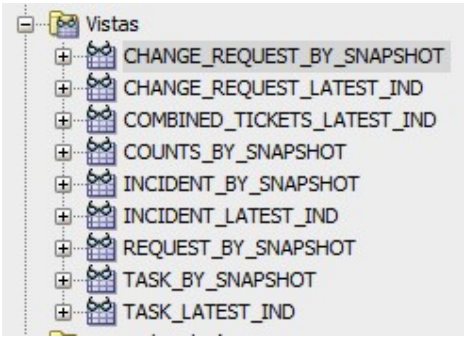


Fig. 2.12 Views in the data warehouse

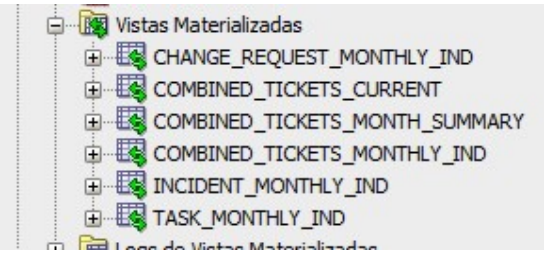


Fig. 2.13 Materialized views in the data warehouse

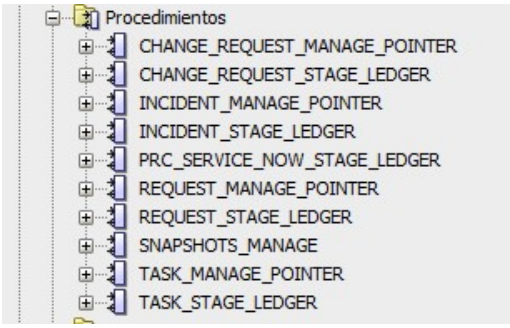


Fig. 2.14 Procedures in the data warehouse

I will explain the most relevant of these objects in the following sections.

2.3.2 Staging tables

Is the step between ODI and the ledger tables. Here is where ODI loads all new tickets or modifications in existing tickets coming from the ServiceNow, it creates a row for each ticket or modification done in any existing ticket and then this data will go to the ledger tables.

PROCESS_TIME	STAGE_TIME	ACTIVE	ACTIVITY_DUE	ADDITIONAL_ASSIGNEE_LIST	APPROVAL	APPROVAL_HISTORY	APPROVAL_SET	ASSIGNED_TO	ASSIGNMENT_GROUP	BUSINESS_DURATION
21/08/20	21/08/20	true	UNKNOWN	(null)	Not Yet Requested	(null)	(null)	Daniel Dalfo	Analytics - SaM	0
22/08/20	22/08/20	true	UNKNOWN	(null)	Approved	(null)	21/08/20	Daniel Dalfo	Analytics - SaM	0

Fig. 2.15 Some fields in Change Request stage table

2.3.3 Ledger tables

Are the main tables that contain all the information from all tickets during the time. All modifications in a ticket are recorded as individual rows in the ledger tables, that means that each time a change to one of the tickets is detected, based on the “Last Updated Field”, it is inserted into the ledger table and the record is assigned a record ID, and the time the record was extracted is also stored in the “Extract Time” field.

These tables are so big, they contain millions of rows, so they are not the most suitable for use in analysis applications due the slow performance.

RECORD_ID	ACTIVE	ACTIVITY_DUE	ADDITIONAL_ASSIGNEE_LIST	APPROVAL	APPROVAL_HISTORY	APPROVAL_SET	ASSIGNED_TO	ASSIGNMENT_GROUP	BACKOUT_PLAN
85044	true	UNKNOWN	(null)	Requested	(null)	(null)	Daniel Dalfo	Analytics - SaM	Revert back to pric
85081	true	UNKNOWN	(null)	Approved	(null)	05/06/19	Daniel Dalfo	Analytics - SaM	Revert back to pric
85077	true	UNKNOWN	(null)	Approved	(null)	05/06/19	Daniel Dalfo	Analytics - SaM	"Revert back to pri
85124	true	UNKNOWN	(null)	Approved	(null)	05/06/19	Daniel Dalfo	Analytics - SaM	"Revert back to pri
85127	true	UNKNOWN	(null)	Approved	(null)	05/06/19	Daniel Dalfo	Analytics - SaM	Revert back to pric
85246	true	UNKNOWN	(null)	Not Yet Requested	(null)	(null)	Daniel Dalfo	Analytics - SaM	Revert back to pric
85278	false	UNKNOWN	(null)	Approved	(null)	05/06/19	Daniel Dalfo	Analytics - SaM	Revert back to pric
85333	false	UNKNOWN	(null)	Approved	(null)	05/06/19	Daniel Dalfo	Analytics - SaM	"Revert back to pri
85347	true	UNKNOWN	(null)	Approved	(null)	14/06/19	Daniel Dalfo	Analytics - SaM	Revert back to pric
85406	false	UNKNOWN	(null)	Approved	(null)	14/06/19	Daniel Dalfo	Analytics - SaM	Revert back to pric
86087	false	UNKNOWN	(null)	Approved	(null)	09/07/19	Daniel Dalfo	Analytics - SaM	Revert back to pric
86872	true	UNKNOWN	(null)	Not Yet Requested	(null)	(null)	Daniel Dalfo	Analytics - SaM	(null)
87431	true	UNKNOWN	(null)	Not Yet Requested	(null)	(null)	Daniel Dalfo	Analytics - SaM	(null)
87473	true	UNKNOWN	(null)	Not Yet Requested	(null)	(null)	Daniel Dalfo	Analytics - SaM	(null)

Fig. 2.16 Some fields in Change Request ledger table

2.3.4 Stage to ledger table procedure

The load procedure that operates on the stage tables to load data into the ledger tables performs the following logic:

1. Look at all rows in stage tables where the field “processed time” is null.
2. For this subset of records, duplicates are eliminated by looking for records with the same record id. If duplicates are found they are considered processed, by updating the “processed time” field.
3. The remainder of the records, where “processed time” is null are loaded into the ledger table.
4. The “processed time” field in the stage table is updated to reflect the time that the load procedure was initiated.
5. The transaction is committed.
6. Records older than 2 weeks old are deleted from the staging table.

At this point the ledger tables will contain multiple rows for a particular ticket.
For example, a single incident might contain 10 rows, each one with a unique record id, and a unique extract date.
A “snapshotting mechanism” detailed in the following sections is described to return the last record for a particular date.

2.3.5 Snapshots table

As mentioned before, for each ticket, there may be more than one record as it could be modified during the time. The purpose of the snapshots table is to have a control on the ServiceNow timeline. This table consists of the following columns:

- **Snapshot**: A date and time corresponding to a “snapshot of interest”. There is a snapshot for the end of each week, the end of each month, and a snapshot for the last time data was loaded.

- **Latest snapshot**: A Y/N flag indicating if this is the current snapshot.
- **Weekly Snapshot**: A Y/N flag indicating that this snapshot represents the end of a week.
- **Monthly snapshot**: A Y/N flag indicating that this snapshot represents the end of a month.

SNAPSHOT_TIME	LATEST_IND	WEEKLY_IND	MONTHLY_IND
31/08/19	N	N	Y
27/10/19	N	Y	N
03/11/19	N	Y	N
01/09/19	N	Y	N
30/09/19	N	N	Y
10/11/19	N	Y	N
01/03/20	N	Y	N
08/12/19	N	Y	N
30/11/19	N	N	Y
13/10/19	N	Y	N
08/09/19	N	Y	N

Fig. 2.17 Snapshots table

2.3.6 Snapshots table procedure

Each time it is run it performs the following logic:

1. Looks for any Sunday in the past (since go live) that does not contain a snapshot at 23:59:59. If no snapshot is found, a snapshot is inserted and flagged as a weekly snapshot.
2. Looks for any month end day in the past that does not contain a snapshot at 23:59:59. If no snapshot exists then one is inserted and flagged as a monthly snapshot.
3. Any snapshot that is not for time 23:59:59 is deleted. This deletes the snapshot that was latest the last time the procedure was run.
4. The current time is inserted as a snapshot and flagged as the latest snapshot.

2.3.7 Pointer tables

For each snapshot, determines the last record ID for each ticket where the extract time is less than or equal to the snapshot time. It stores a single record ID for each snapshot and each ticket, and this allows the view of the tickets on a timeline.

RECORD_ID	SNAPSHOT_TIME
91067	30/04/20
90967	30/04/20
90934	30/04/20
90928	30/04/20
90909	30/04/20
90912	30/04/20
90913	30/04/20
90917	30/04/20
90819	30/04/20
90828	30/04/20
90855	30/04/20
90783	30/04/20
90786	30/04/20
90792	30/04/20

Fig. 2.18 Change Request pointer table

2.3.8 Pointer tables procedure

This procedure performs the following logic:
Looks for any snapshot times in the pointer table, that do not exist in the snapshot table and deletes them. This caters for the latest snapshot which will change each time the logic is run.
For each snapshot in the snapshots table, that does not exist in the pointer table – calculate the max extract time per ticket, with the corresponding record ID, and insert the snapshot time, and record id into the pointer table.

2.3.9 By snapshot views

These views are created combining the ledger tables with the snapshots table and the pointer tables. The reason for this is to have all the information from the ledger tables and being able to filter the data according to the snapshots.
For example filtering on Latest snapshot = Y' then it will return the state of the ServiceNow ticket as it was when the extract was last run.

```
SELECT *
FROM service_now.change_request_by_snapshot
WHERE assigned_to = 'Daniel Dalfo'
AND LATEST_IND    = 'Y' ;
```

sultado de la Consulta

SQL | Todas las Filas Recuperadas: 11 en 1,135 segundos

	SNAPSHOT_TIME	LATEST_IND	WEEKLY_IND	MONTHLY_IND	COUNT	RECORD_ID	ACTIVE	ACTIVITY_DUE	ADDITIONAL_ASSIGNEE_LIST	APP
1	22/08/20	Y	(null)	(null)	1	89785	false	UNKNOWN	(null)	Appr
2	22/08/20	Y	(null)	(null)	1	95578	true	(null)	(null)	not
3	22/08/20	Y	(null)	(null)	1	86087	false	UNKNOWN	(null)	Appr
4	22/08/20	Y	(null)	(null)	1	89062	false	UNKNOWN	(null)	Appr
5	22/08/20	Y	(null)	(null)	1	99793	true	UNKNOWN	(null)	Appr
6	22/08/20	Y	(null)	(null)	1	95896	false	(null)	(null)	appr
7	22/08/20	Y	(null)	(null)	1	97518	true	UNKNOWN	(null)	Not
8	22/08/20	Y	(null)	(null)	1	95838	true	(null)	(null)	not

Fig. 2.19 Change Request by snapshot table filtered

2.3.10 Combined materialized views

Saving all this data has a final purpose that is to analyze behaviours and trends of the teams in the company, so it is interesting to have objects with data from all kinds of tickets, not only individual tables for each kind. To analyze the data, many times we don't need all time data, sometimes it is enough just to use the latest status of the tickets or to have just a monthly view.

For these reasons we created three materialized views that combines different tables and filters to achieve these results:

- **Combined Tickets Current**: Combines the data from the Change Request by Snapshot, Incident by Snapshot, and Task by Snapshot tables, filtered by the latest snapshot. Provides a current view of the tickets in the ServiceNow.
- **Combined Tickets Monthly Ind**: Combines the data from the Change Request by Snapshot, Incident by Snapshot, and Task by Snapshot tables, filtered by the monthly snapshot. Provides a monthly view of the tickets in the ServiceNow.
- **Combined Tickets Monthly Summary**: Has the same logic as the previous but less columns, to achieve a faster performance when analyzing.

2.4 Data Module design

Now we have the information being loaded and stored in the data warehouse in a daily base, rows and rows of information that are extracted everyday from the ServiceNow and stored in different tables. That is a lot of information and many times, the person who wants to analyze this data doesn't need to see all details for all tickets.

In the Data Module we will prepare and organize these tables to later display the data on dashboards in an understandable and synthetic way.

2.4.1 Architecture

I pulled all ServiceNow tables from the data warehouse into the Data Module, and as you can see in Fig. 2.20 I organized them in separate folders according to their use.

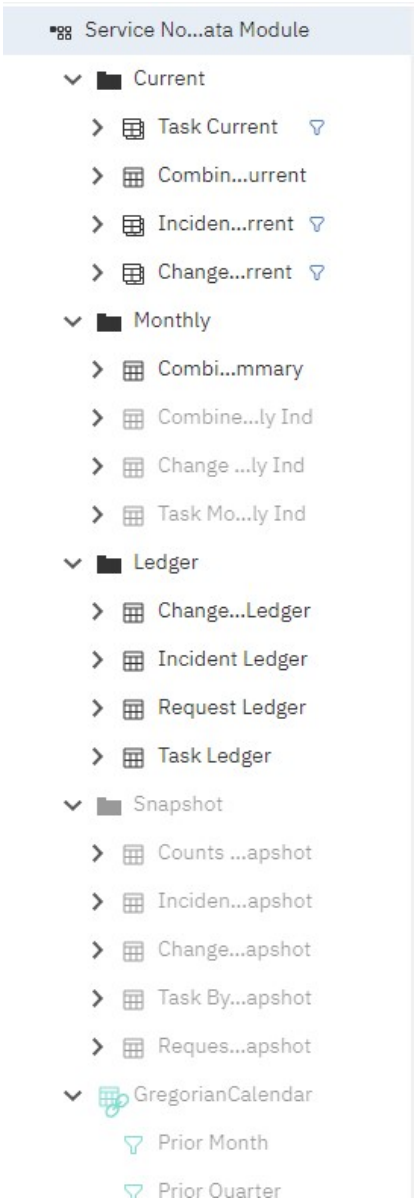


Fig. 2.20 Architecture of the Data Module

In the current folder I stored the Combined Tickets Current materialized view and three more views created in the Data Module that I will explain in the following sections. So the current folder stores the tables that contain data from the current state of ServiceNow.

The monthly folder contains the tables that contain data from the monthly snapshots of the ServiceNow. This data is useful to analyze the ServiceNow state month by month.

At the ledger folder I stored the main tables that contain all the data, these folders are not directly used for analysis in this project but can be used in the future for detailed analysis.

2.4.2 New views

One of the possibilities of the Data Module is to create new tables by filtering or mixing the existing ones. As you can see in Fig. 2.21 from the Combined Ticket Current I created 3 different views, one for each ticket type, to have the possibility to analyze the trends of them separated.

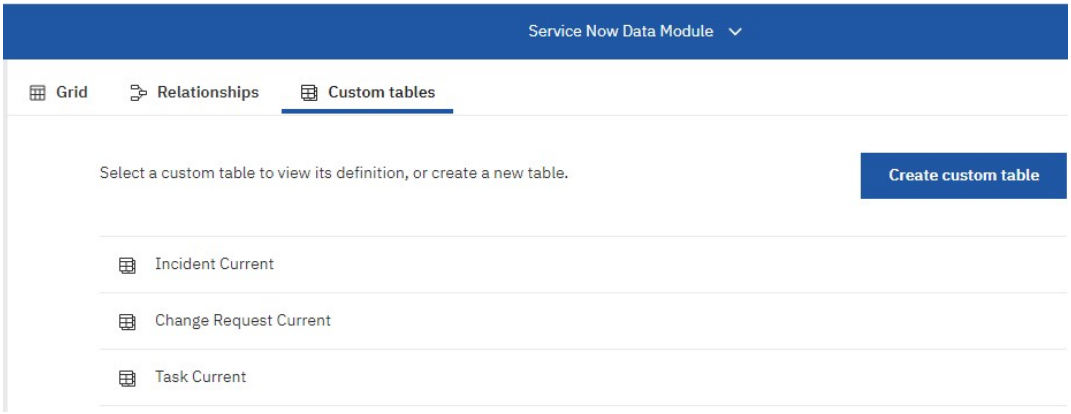


Fig. 2.21 New views created in the Data Module

2.4.4 Filters

The filters are a very interesting feature of Data Module, these are stored in the tables like columns but instead of information they contain some rules.

This definition can be a bit confusing but now I will explain some examples and it will be much clearer. For all the tables I created the filters “Open Tickets” and “Closed Tickets”.

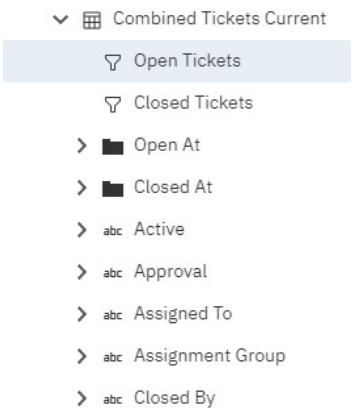


Fig. 2.22 “Open Tickets” and “Closed Tickets” filters for table CTC

When using the dashboards tools that I will explain in the following sections, if you drag one of these filters on top of a chart it will impose the rules that have been previously defined. For example, if you drag the “Open Tickets” filter on top of a graph showing the tickets for a user, then the graph will only show the open tickets for the user. The filters are defined using a coding language by Cognos very similar to SQL.

2.4.5 Calculations

One of the most interesting features of Cognos Data Module is the possibility to create new columns in the tables that the value of their fields is a calculation defined by the user. Common calculations can be mathematical calculations, to join the values from different columns in a single one, extract only a part of the data in a single field, etc…

I used this feature to display the time in different ways. In our data warehouse there is a column named “Opened_At” and another one named “Closed_At”, both contain the year, month, day, and even the hour when the ticket was opened or closed, but sometimes we will want to group the tickets only by year, or filter them by month, so to do that, I created the calculations shown in Fig. 2.23.

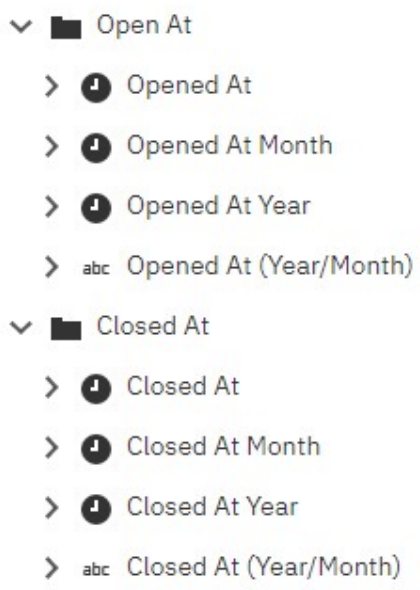


Fig. 2.23 Calculations on Data Module

Fig 2.24 shows some examples of records for the column “Opened At” and the calculations that I created around this column, each of them is used for a different purpose that will be seen in the following sections.

Opened At	Opened At Month	Opened At Year	Opened At ...ar/Month)
2015-09-10T14:14:22	09	2015	2015/09
2015-09-14T09:02:13	09	2015	2015/09
2015-09-14T10:21:21	09	2015	2015/09
2015-09-14T10:35:58	09	2015	2015/09
2015-09-14T13:40:56	09	2015	2015/09
2015-09-14T17:24:49	09	2015	2015/09

Fig. 2.24 Opened At calculations

2.4.6 Navigation paths

A navigation path is a collection of columns that users might associate for data exploration. This will be useful in the following steps when using the dashboards, users can drill down and back to change the focus of their analysis by moving between levels of information.

I created navigation paths using the time calculations mentioned in the previous section. I called these navigation paths Time Range, and they will be useful when using dashboards to change the way a graph is displaying the time. Fig. 2.25 is showing an example of one the navigation paths I created in the table Combined Tickets Current with the Opened At calculations.

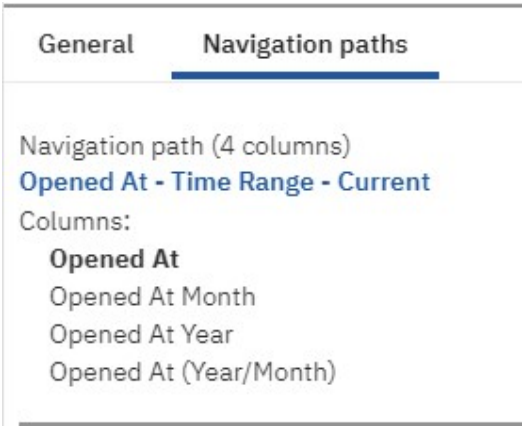


Fig. 2.25 Opened At - Time Range - Current navigation path

2.4.7 Gregorian calendar

This is an external table that I added to the data module but it is not coming from the data warehouse. This table is stored inside Cognos and was previously configured by an external person in the company.

As you can see in Fig. 2.26 the Gregorian Calendar table contains different filters relationed with time.

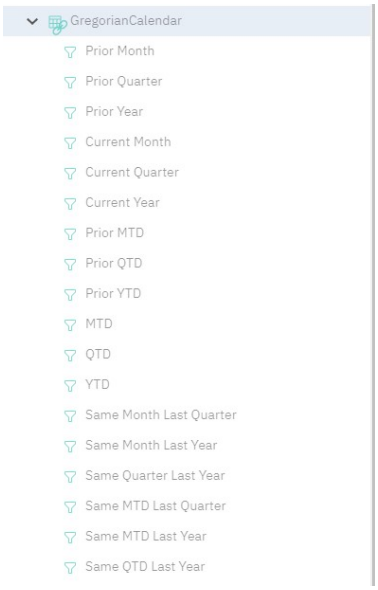


Fig. 2.26 Gregorian Calendar filters

Data module has the option to link the Gregorian Calendar table with columns from other tables and then use these filters applied to the linked column.

For example I linked the Gregorian Calendar table to the Open At columns from my tables.

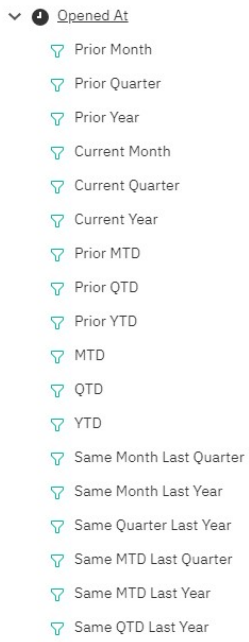


Fig. 2.27 Opened At linked to the Gregorian Calendar filters

This gives me the possibility to apply the Gregorian Calendar filters to this field when using Dashboards. If I create a KPI that is displaying the open tickets for an user, if I drop the Prior Month filter on top of the KPI it will only show the open tickets for that user in the prior month.

2.4.8 Data groups

This is a very interesting feature from Cognos Data Module. You can organize the data from a certain column into custom groups so that the data is easier to read and analyze.

In the company there are different teams that work on related issues, like for example applications. I created a data group called IT Teams that groups specific teams from the column Assignment Group into bigger teams.

Assignment Group	IT Teams
Application E1 CNC Operations	Application
Application E1 Security Champions Finance	Application
Null	Other Teams
Null	Other Teams
ITSM Admin	Other Teams

Fig. 2.28 IT Teams example

As seen in Fig. 2.28 different teams like “Application E1 CNC Operations” and “Application E1 Security Champions Finance” in the column IT Teams are grouped under the same team “Application”. At the time that I took the screenshot Fig. 2.28 the teams that didn’t have an IT Team defined were under “Other Teams”.

2.5 Dashboards design

At this point, the data is prepared to be displayed in an easy and understandable way so an user will be able to analyse different factors and trends about the ServiceNow. This section will show some dashboard designs that I created to display the data.

2.5.1 ServiceNow Overview By Month

This first dashboard will show an overview of the ServiceNow state month by month. It is built using the Combined Tickets Month Summary, as I mentioned in previous sections, this table is a materialized view that contains data from the ledger tables filtered by the monthly snapshots, so it provides a monthly view of the state of the ServiceNow.

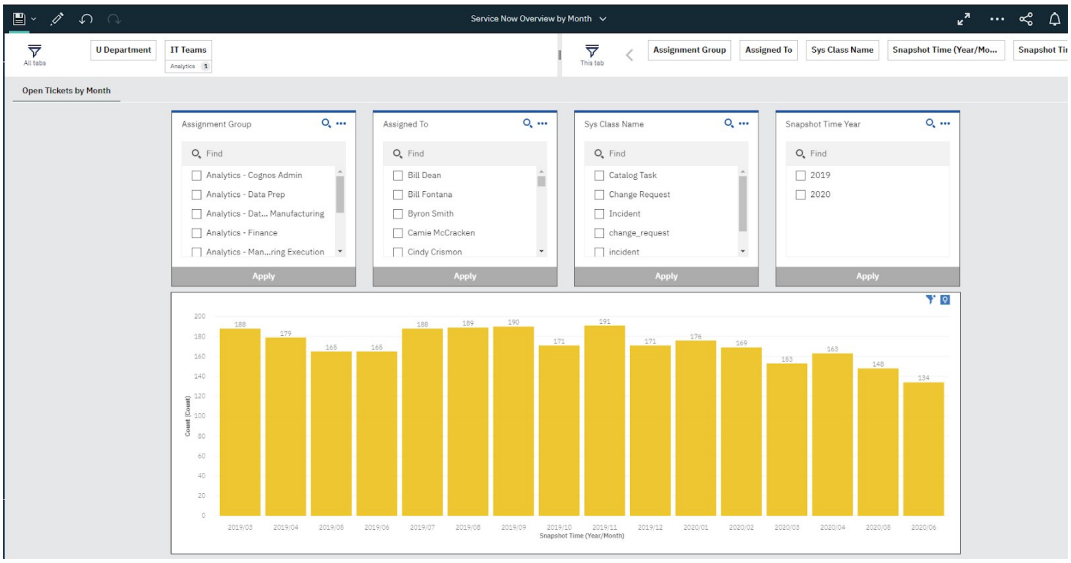


Fig. 2.29 ServiceNow Overview By Month dashboard

The dashboard is composed of a column graph, four checklists, and some filters on top of the screen.

As you can see in Fig. 2.30, the graph is showing the number of tickets that were open on ServiceNow during the time.

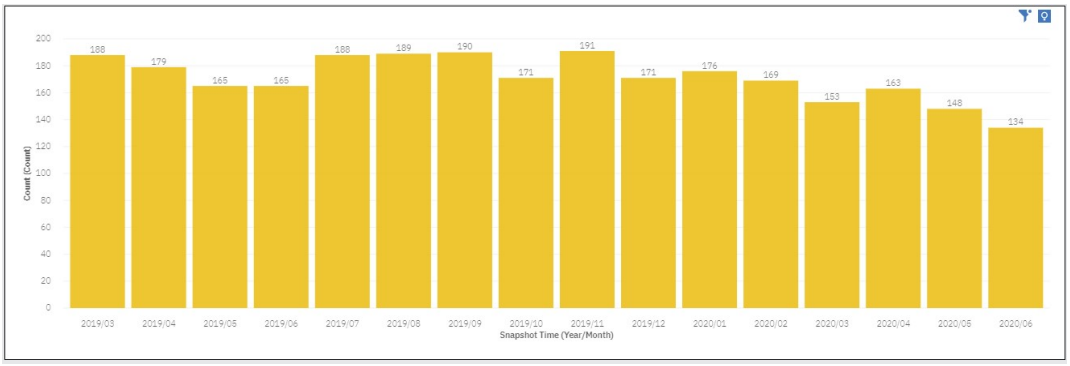


Fig. 2.30 ServiceNow Overview By Month column graph

The time range can be changed using the navigation path that I explained in section 2.4.6, we can select seeing the open tickets by month, by year and month, by year, or even by day.

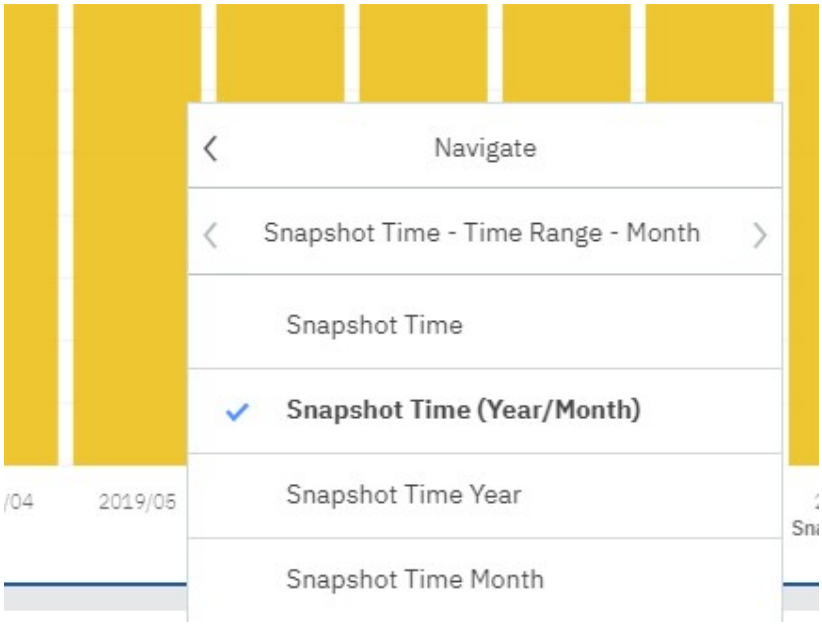


Fig. 2.31 Navigation path on a dashboard

The checklist displays different attributes that if selected will filter the data displayed in the graph.

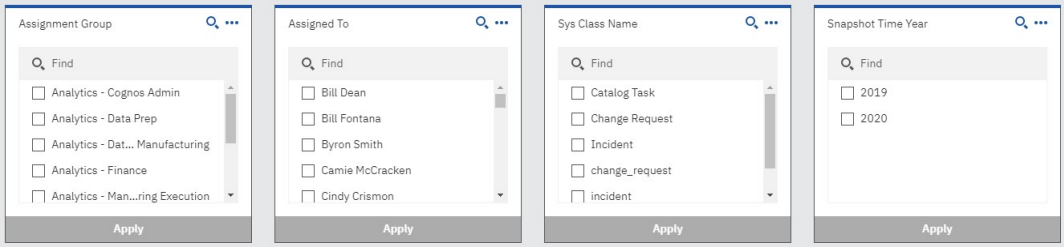


Fig. 2.32 Filter checklists on the dashboard

This is very useful if you want to see the open tickets by a certain team, by an user, you can select the ticket type and also the year. When selecting one option on the checklists and clicking “Apply” it will filter the data displayed on the graph and on the other checklists.

In the top of the page, there are more filters such as the department of the company, another level of teams or you can even select individual month and year, as shown in Fig. 2.33, to compare the same month in different years.

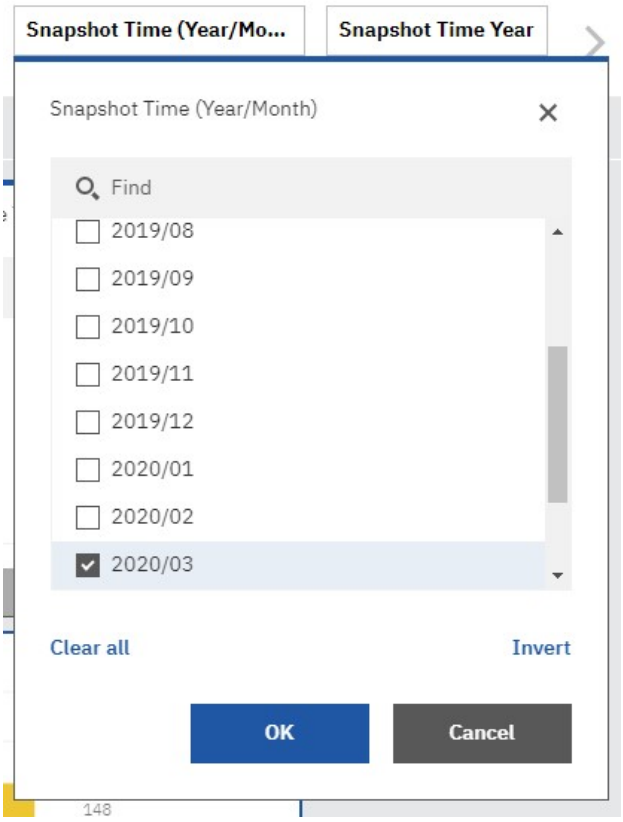


Fig. 2.33 Filter by individual year/month

Fig. 2.34 shows the graph filtered by March on 2019 and 2020.

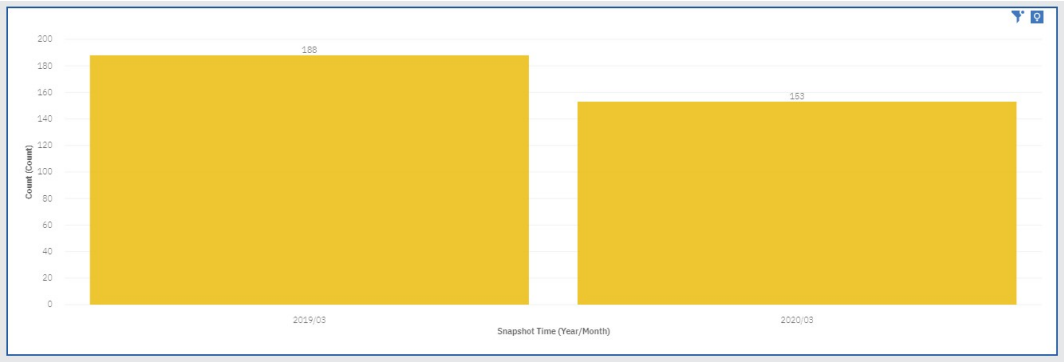


Fig. 2.34 Filtered graph

2.5.2 Tickets Trend dashboard

This dashboard is more complex and displays data from the Combined Tickets Current table, so that means that we will see the data in the current state of ServiceNow. As shown in Fig. 2.35, this dashboard has more tabs that I will explain in detail one by one.

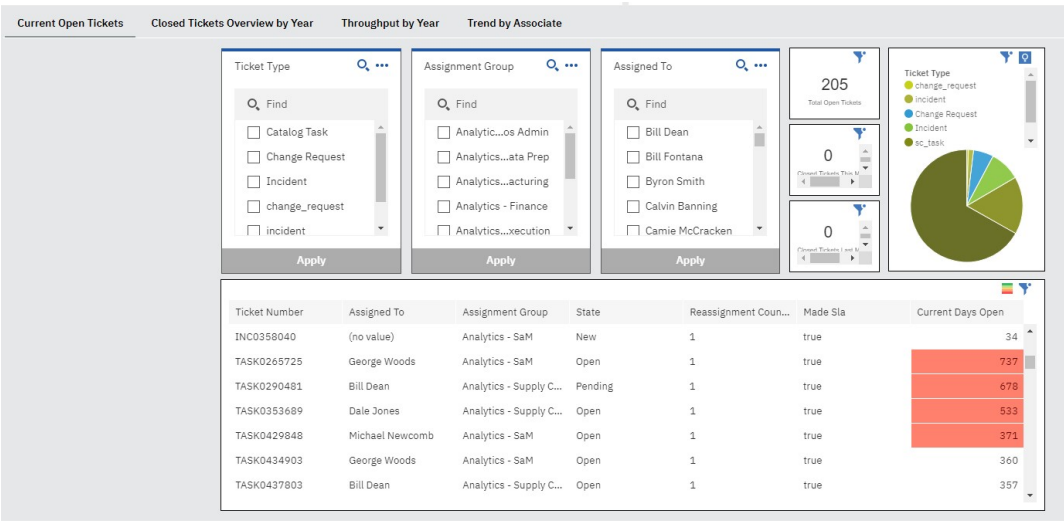


Fig. 2.35 Tickets Trend dashboard

First tab is called Current Open Tickets, it provides information about the tickets that are open in the actual moment and has the possibility to filter by ticket type, department (IT Teams), team (Assignment Group), and user (Assigned To). The way to filter the data is by checklists as we have seen in the previous dashboard, it also has a section with 3 KPI showing the total open tickets at the

moment, the tickets that were closed this month, and the tickets that were closed last month. There is also a pie graph that displays the % of tickets by ticket type.

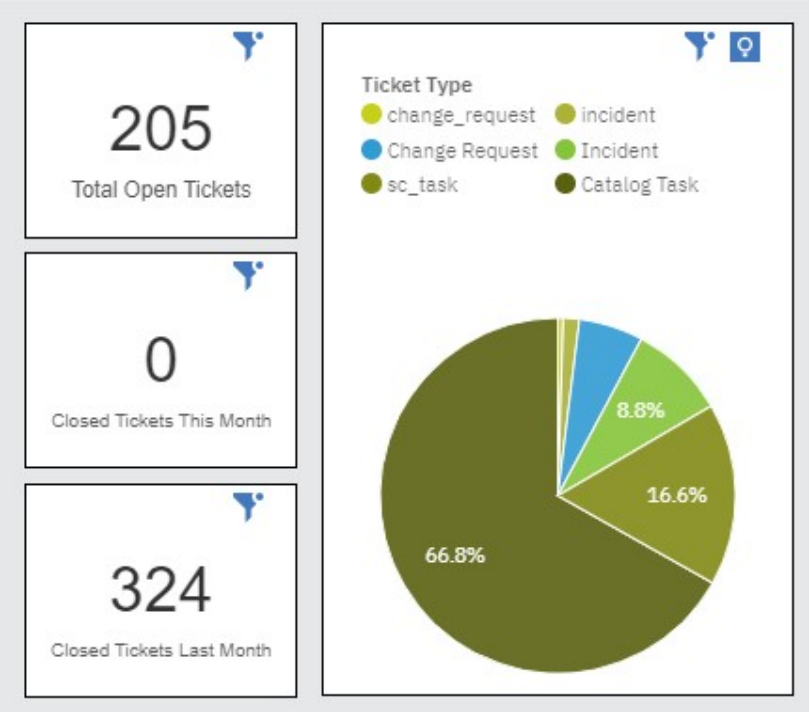
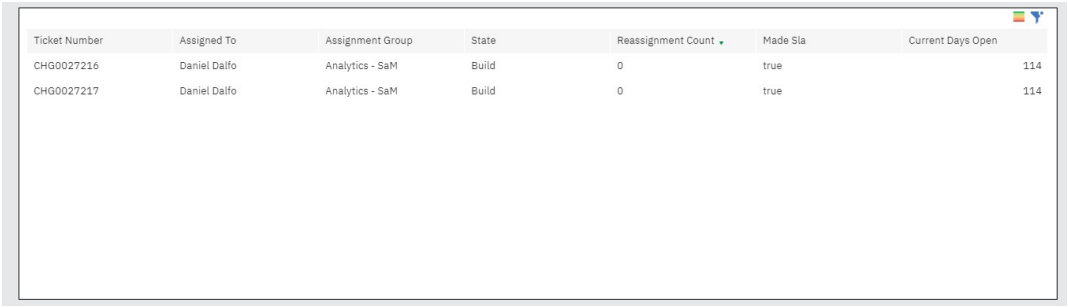


Fig. 2.36 KPI's and pie graph

There is a table showing basic data from the individual open tickets at the moment and when a ticket is opened for more than a year it is coloured in red. The data displayed in the table can be filtered using the checklists, Fig. 2.37 shows the table showing data filtered by my user name and ticket type equal to "Change Request".



Ticket Number	Assigned To	Assignment Group	State	Reassignment Count	Made Sla	Current Days Open
CHG0027216	Daniel Dalfo	Analytics - SaM	Build	0	true	114
CHG0027217	Daniel Dalfo	Analytics - SaM	Build	0	true	114

Fig. 2.37 Dashboard table filtered

Second tab is called Closed Tickets Overview by Year, the purpose of this tab is to analyse the trends regarding closing tickets.

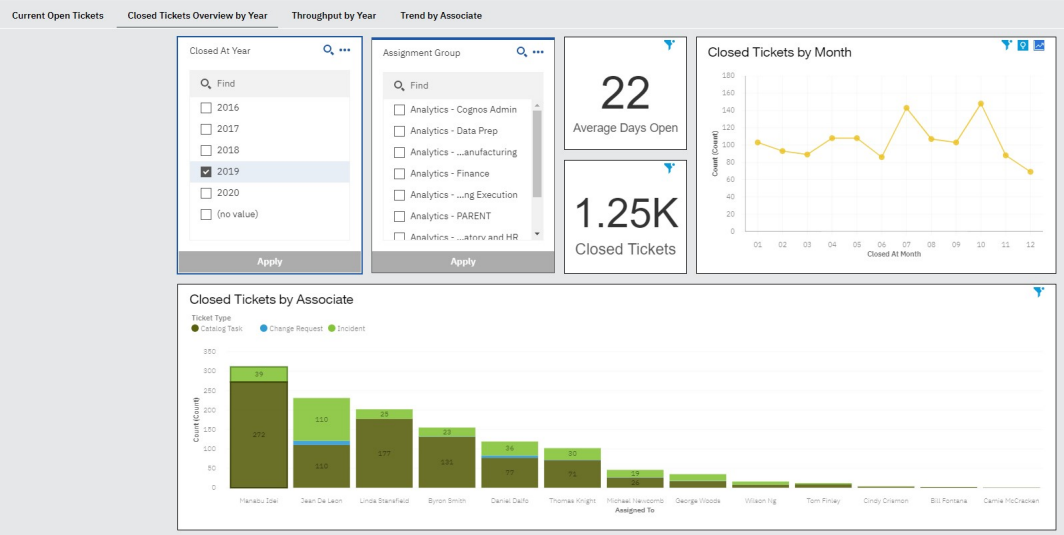


Fig. 2.38 Closed Tickets Overview by Year tab

Usually this tab will be used by team leaders who want to analyse the trends in their team, for this reason the filters on this tab are year and assignment group (team). To the left of the filters we find a couple of KPIs that will show the average of days it takes to close a ticket and the tickets that the team closed that year. Fig 2.39 shows the next graph Closed Tickets by Month, here we can clearly see the trend on closing tickets per year in a very visual way.

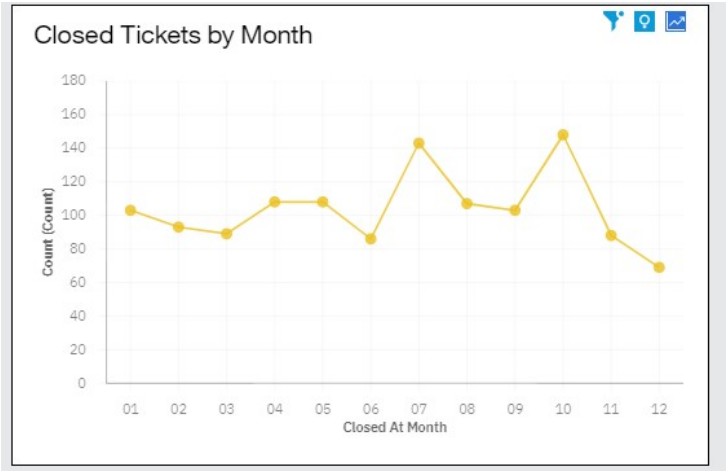


Fig. 2.39 Closed Tickets by Month graph

Below these items we find a stacked column graph that I called Closed Tickets by Associate, it is useful to see how many and what kind of tickets closed each user in the team.

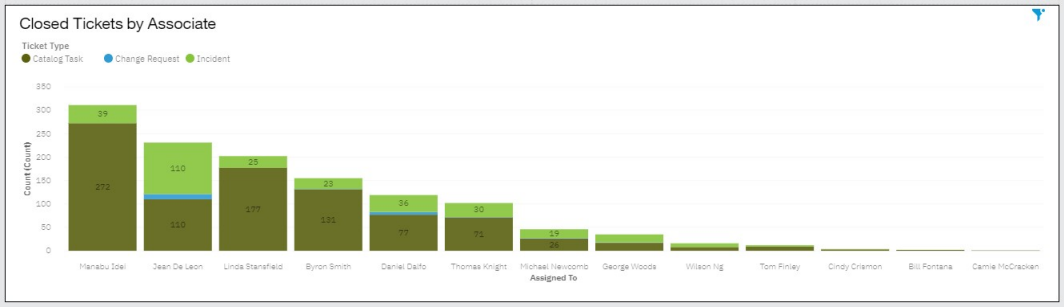


Fig. 2.40 Closed Tickets by Associate graph

A very interesting feature of Dashboards in Cognos is that if you click on the name of one of the users in the graph, the other visualizations will be filtered to show data only for the selected user, so it is very interactive.

The next tab is showing a comparison during the time between the tickets that were opened and the tickets that were closed.

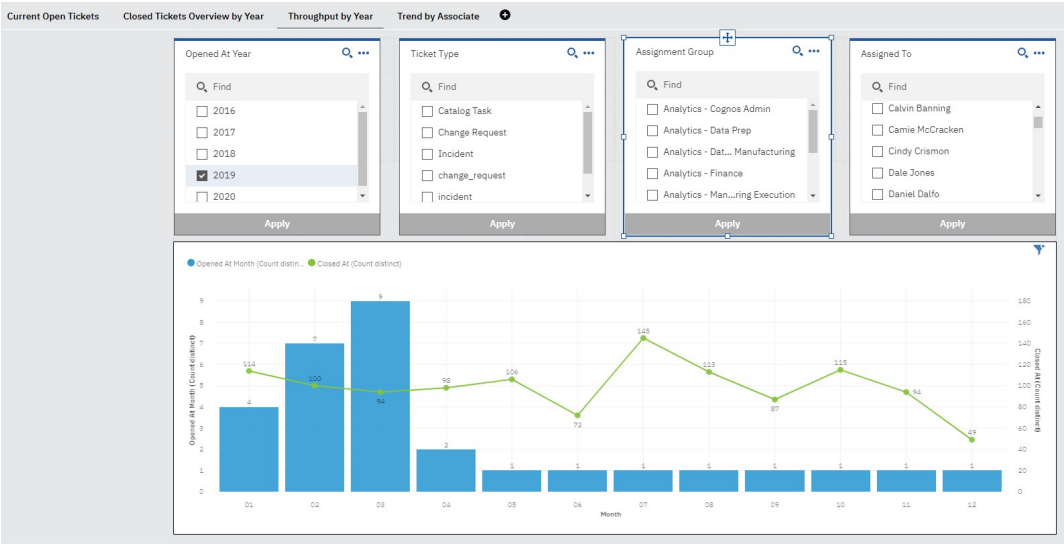


Fig. 2.41 Throughput by Year tab

This tap shows the performance of the teams, the blue columns show how many tickets were opened and the green line how many tickets were closed that month. Using the Throughput by Year tab a team leader can see if they need more people in the team because they don't close enough tickets or if the trend is good.

The fourth tab is called Trend by Associate and as it is visible in the name it provides tools to analyze the performance of individual users.

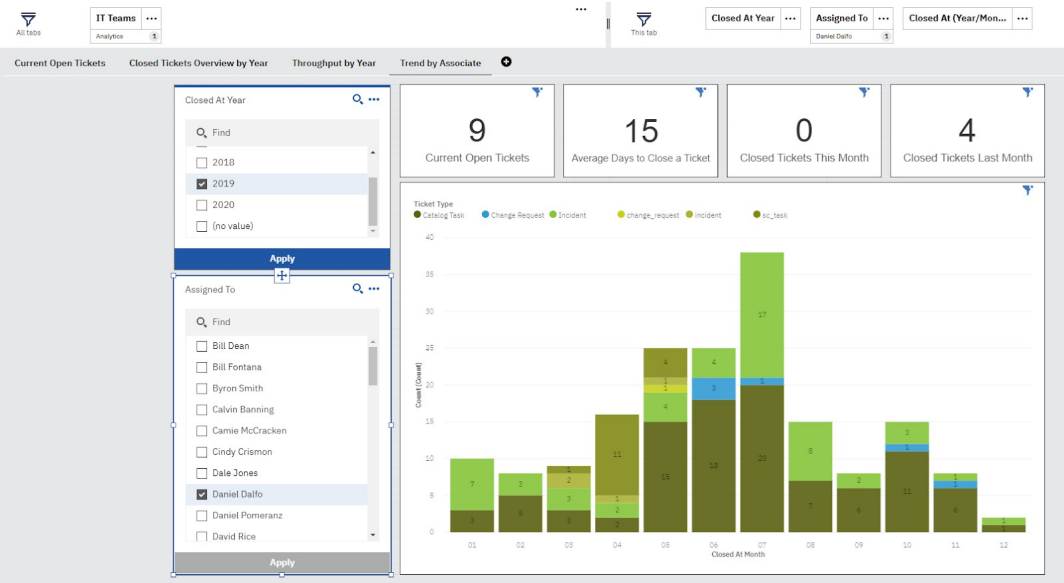


Fig. 2.42 Trend by Associate tab

On the left side we can filter by year and by user. In Fig. 2.43 we can see a zoom to the KPIs of this tab, they are displaying the number of open tickets at the moment, the average days that the user needs to close a ticket, the number of tickets closed this month, and the number of tickets closed last month.

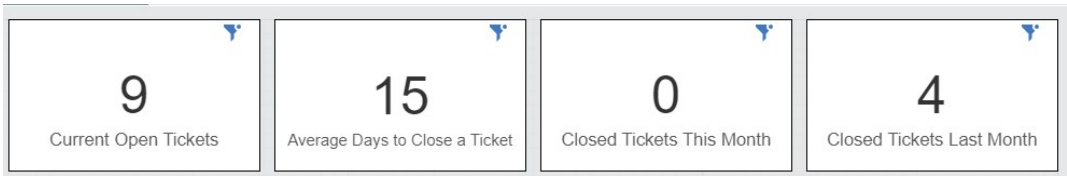


Fig. 2.43 KPIs on Trend by Associate tab

Under the KPIs there is the main part of the tab, a stacked column graph showing the number and type of tickets closed by month.

One interesting feature on this tab is to change the time range on the graphic, using the navigation path, mentioned in previous sections, to year / month as shown in Fig. 2.44.

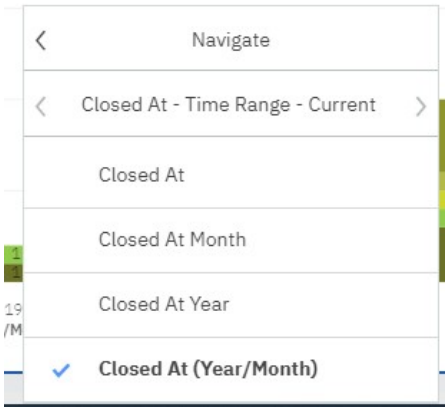


Fig. 2.44 Closed At navigation path

After changing the time range, using the filters we can select the same month for 2 different years and compare the performance. Fig. 2.45 shows an example filtering by my user name (Daniel Dalfó) and closed at May 2019 and May 2020.

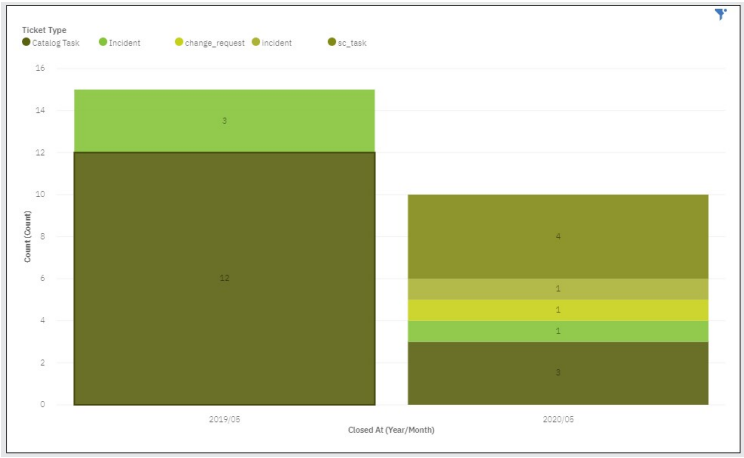


Fig. 2.45 Graph on Trend by Associate tab

Chapter 3. Implementation, issues and results

In this chapter I will explain how I implemented each part showing some key steps on the creation of the modules, I will also explain some issues that I had during the process and how I solved them, and I will also talk about things that I learned based on trial and error.

3.1 ODI implementation

As we have different ticket types and each type records different information, I had to create a process for each type : Incident, Change Request, and Task. The process for each type of ticket is quite similar so in this chapter I will only explain the example for the Change Request.

3.1.1 Creating the links

The first thing to do is create the topology of the system. There has to be a connection from the ServiceNow to ODI, and a connection from ODI to the data warehouse.

The connection from the ServiceNow to ODI is made through a JDBC driver from CDATA, a software provider of data access and connectivity solutions. They provide a driver and an URL that have to be inserted in ODI to make the connection to ServiceNow.

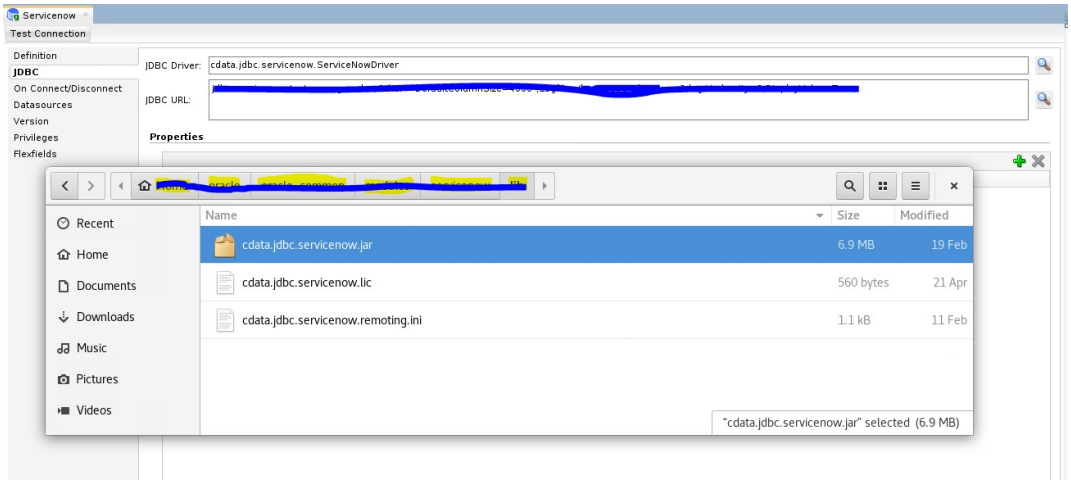


Fig. 3.1 Connection from ServiceNow to ODI using CDATA driver

The connection between ODI and the data warehouse is done by a similar process but in this case the JDBC driver is provided by Oracle.

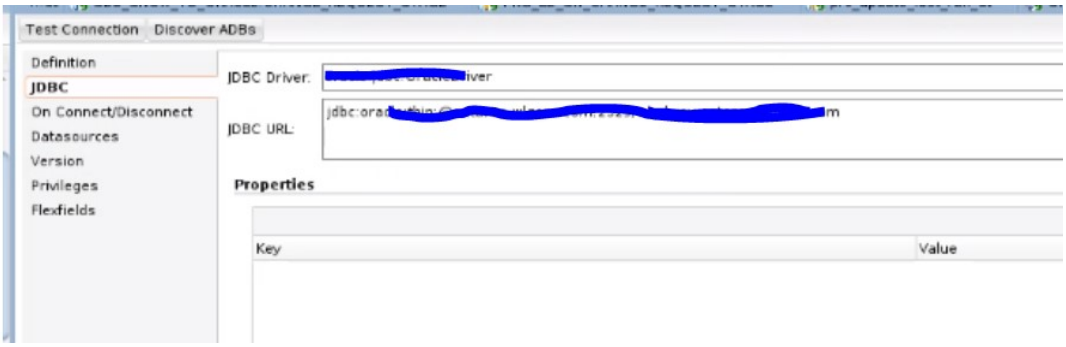


Fig. 3.2 Connection from ODI to the data warehouse using Oracle driver

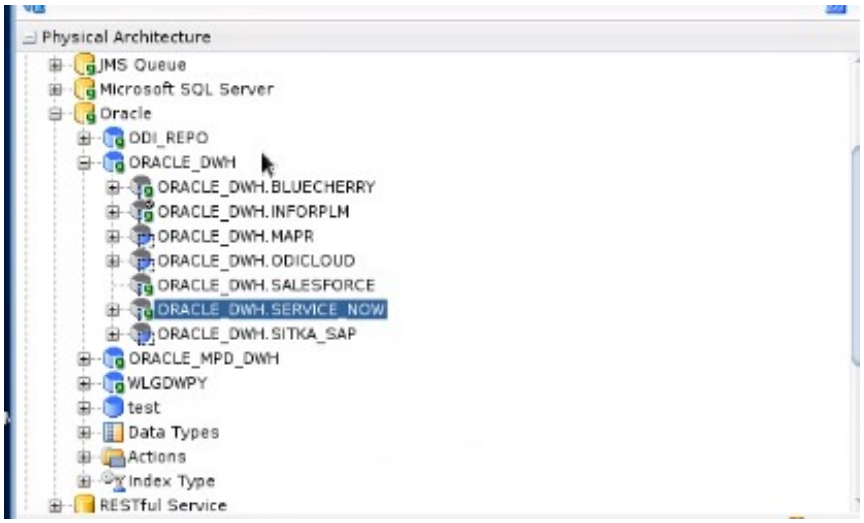


Fig. 3.3 ServiceNow schema in the data warehouse

3.1.2 Creating the models

Now we have to define how the tables where we want to store the data from ServiceNow to the data warehouse will look like. To do so, ODI provides an interface to define the models with all the columns and their attributes, for each column we will have to define the name, the type of data that will be stored there, the maximum length of the data, if it can be empty or not, etc… For each ticket ServiceNow stores a lot of columns, and that would be a lot of work to define every single column for each model...

ODI provides a very useful tool called Reverse Engineer that will copy the tables from the data source and create a model. For each model I used the Reverse Engineer tool and then modified some attributes if it was necessary.

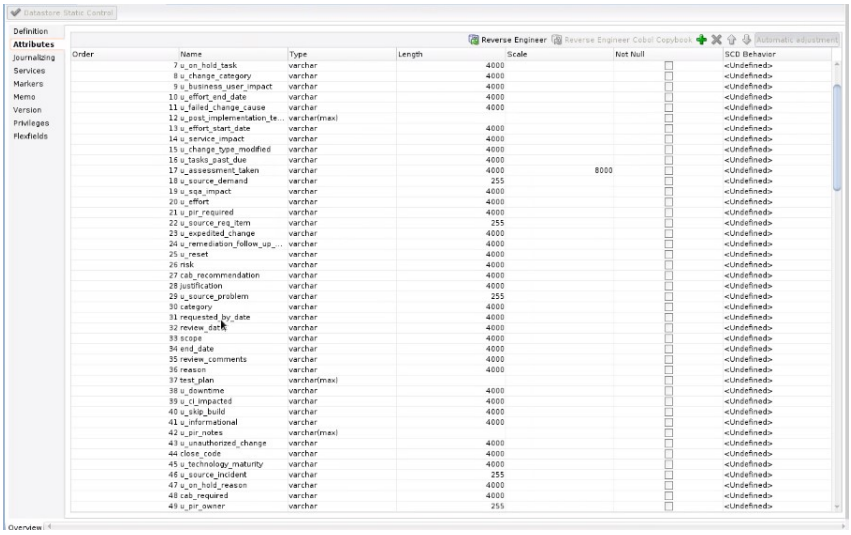


Fig. 3.4 Change Request model in ODI

3.1.3 Creating the mappings

Having all the models, now is time to create the mappings between the tables in the ServiceNow and the tables in the data warehouse.

Fig 3.5 shows the interface to create mappings, the object on the left side is the table from the ServiceNow and the object on the right side is the model that we previously created.

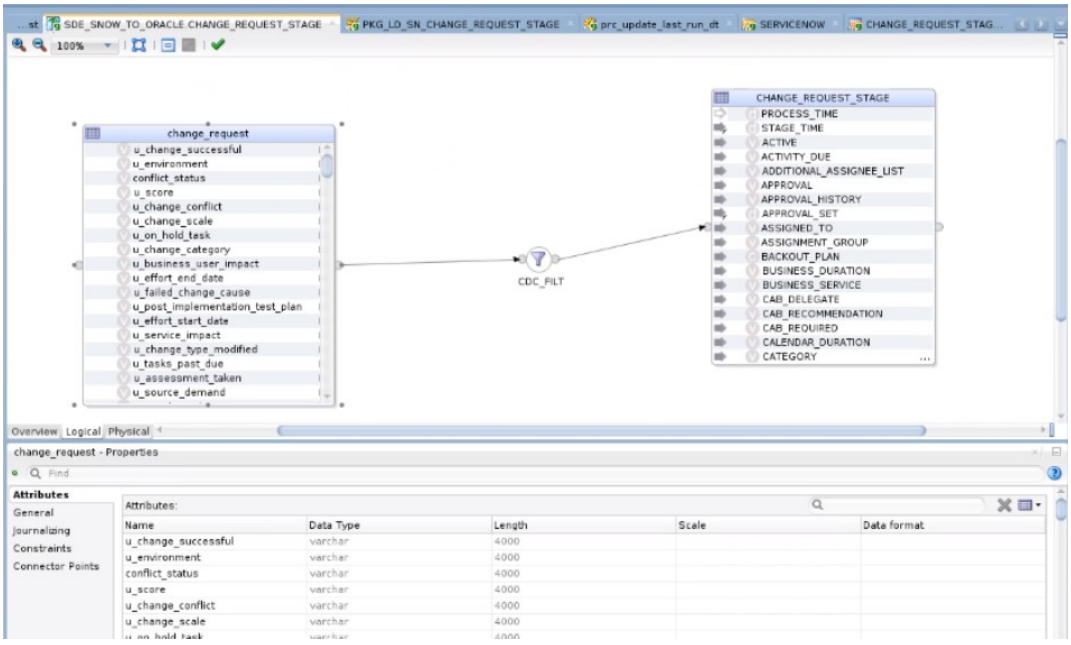


Fig. 3.5 Change Request mapping

In between the models there is a filter, ODI provides some tools and components to generate logical orders when processing the mappings. In this case it is a simple filter called CDC_FILT that creates a condition using two variables that we previously created in ODI. Fig. 3.6 shows the logic that CDC_FILT filter will perform when running the mapping.

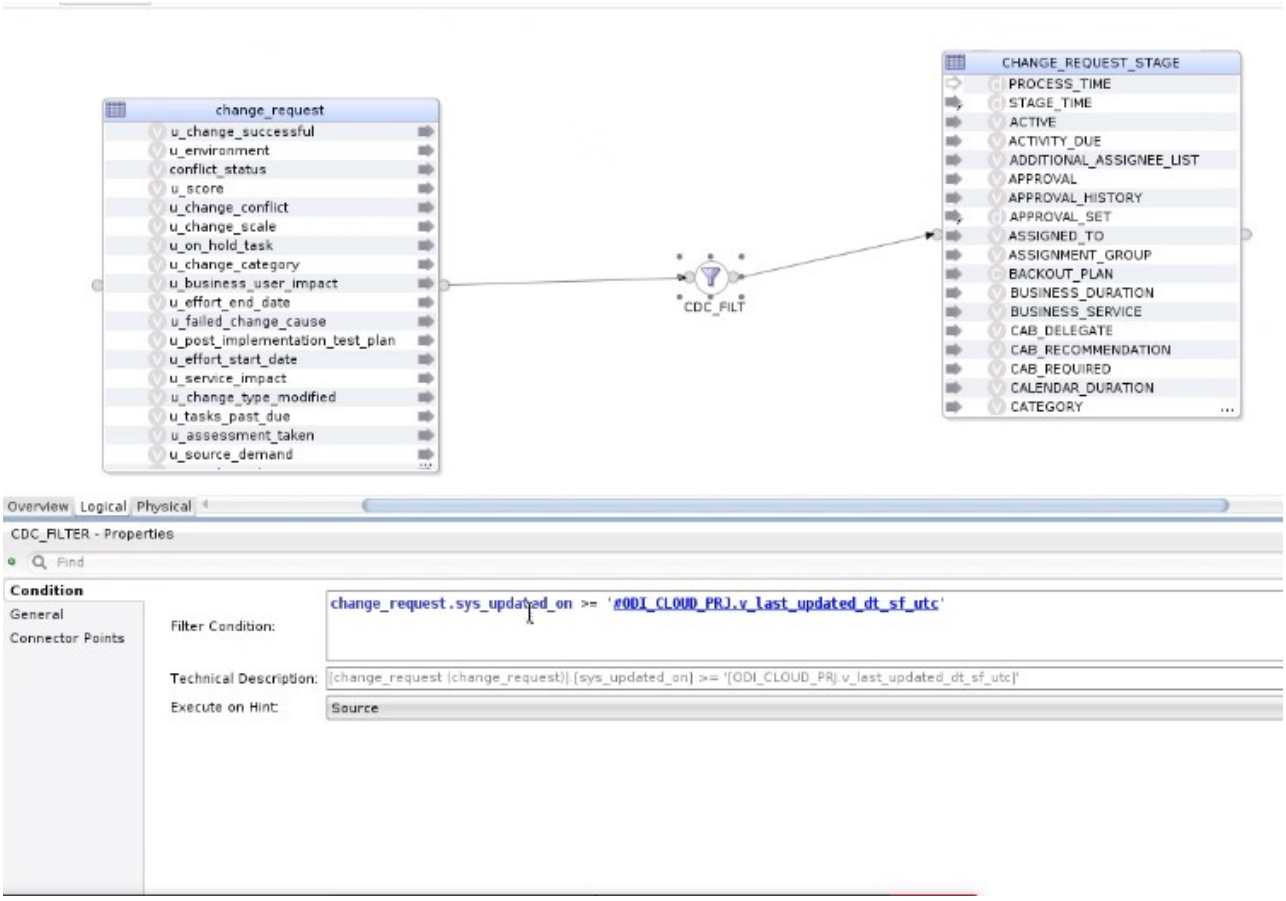


Fig. 3.6 CDC_FILT filter logic

These variables are representing time. “change_request.sys_updated_on” records the time when the Change Request table in the ServiceNow is updated, the symbols “>=” means greater or equal than, and “#ODI_CLOUD_PRJ.v_last_updated_dt_sf_utc” records the time when the data warehouse is updated by the ODI . So that means that the mapping will be done only if the ServiceNow table was updated more recently than the data warehouse table to avoid wasting time and computational capacity if there isn’t new data.

3.1.4 Creating the packages

At this point everything is ready and it just has to be assembled. Fig. 3.7 shows the steps on the path that we defined for the Change Request, each object in

the path describes a process that has some logic inside. As you can see, after object number 4 our path forks, the system will choose one path or another depending on whether process 4 ran successfully or something went wrong.

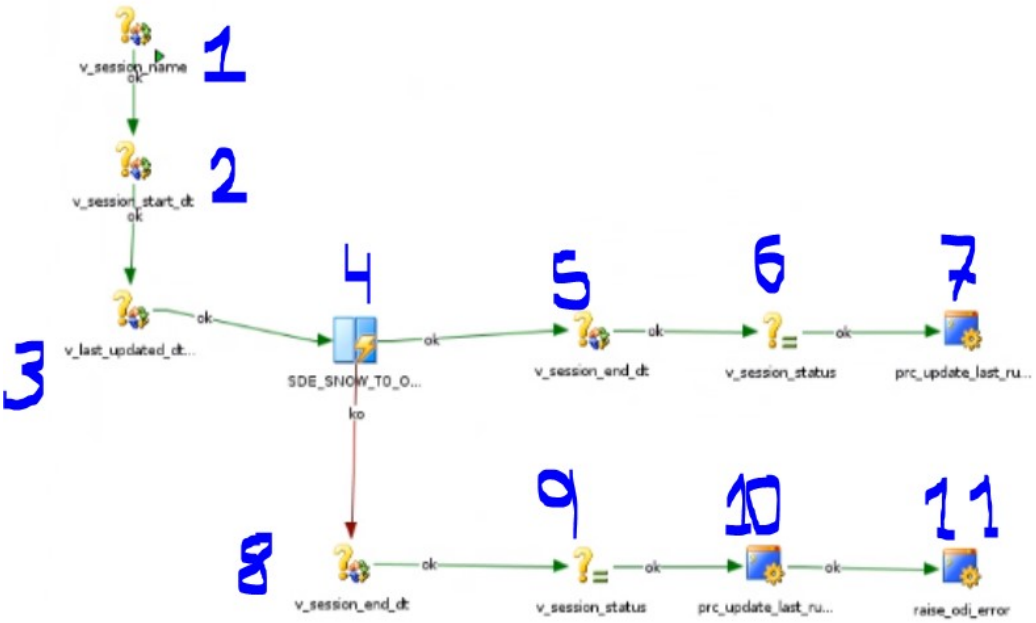


Fig. 3.7 Change Request package step by step

I will now explain the logic of the package step by step:

1. Records the name of the session in a variable. This is used to have a list of the sessions runed in the database.
2. Record the time when the session starts.
3. Make an SQL query to a table called “ODI_ADMIN_VARIABLES” in the data warehouse where we record some data of each loading session, and pull the time of the last successful session.

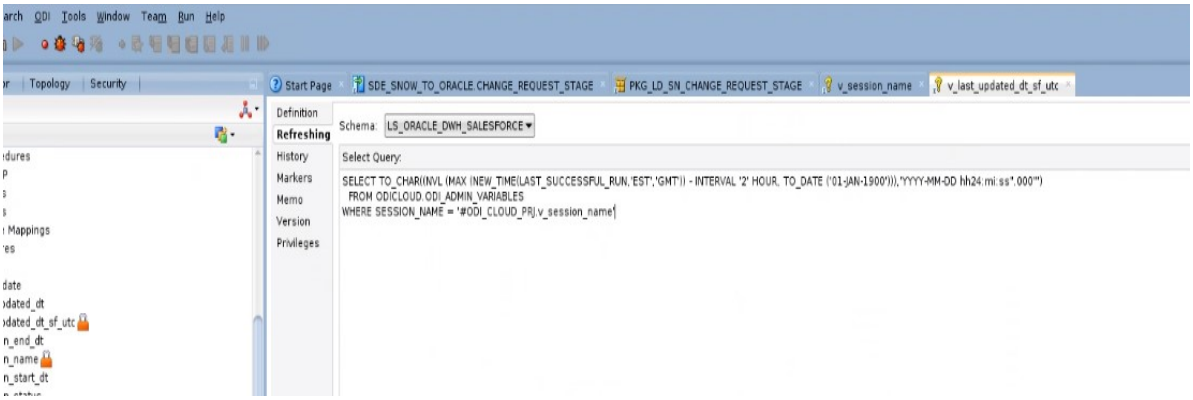


Fig. 3.8 SQL query pulling the last time running the package

4. This step runs the mapping that I explained in the previous section. The filter in the mapping is now using the variable pulled in the step before. It will only download the new data from that time till now. This way of extracting only the new data from a certain time is called incremental update.

At this point, if the mapping has run successfully, the process will continue through step 5 and if something goes wrong it will go to step 8.

5. Captures the end time of the session.
6. Captures the status of the session, in this case was successful.
7. Runs an SQL procedure that records the name of the session, the end time and the status in the "ODI_ADMIN_VARIABLES" mentioned before.
8. Captures the end time of the session.
9. Captures the status of the session, in this case failed.
10. Runs the same SQL procedure as in the step 7, in this case it will record a row with the end time, a failed status and the name of the session.
11. Records the error in a log document.

3.1.5 Creating the procedure

For the ServiceNow system we have a procedure called "SN_TO_ORCL_DAILY" that runs the packages for each ticket type. As shown in Fig. 3.9 every package is a step in the procedure.

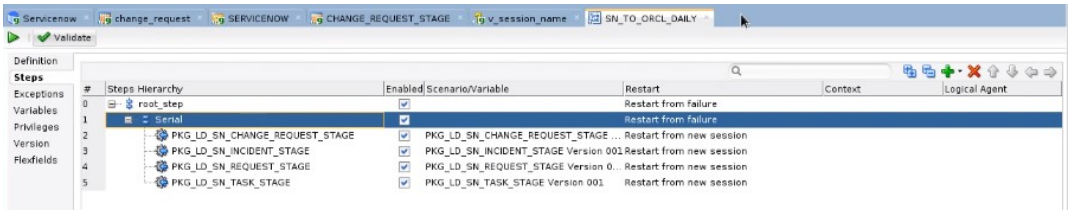


Fig. 3.9 ODI procedure to run all ServiceNow packages

We have created an automated exception that will send an email if something goes wrong during the procedure.

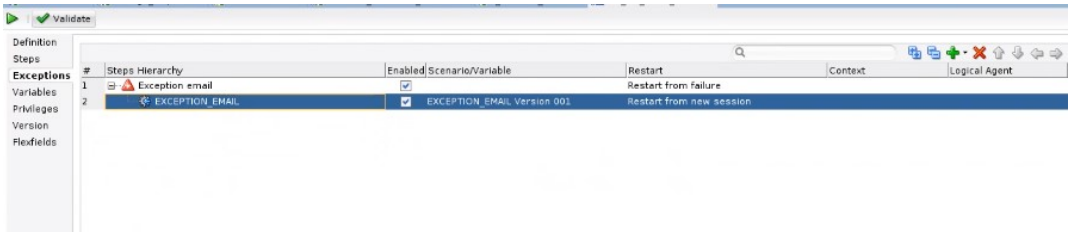


Fig. 3.10 Exception in the procedure

This process is running everyday by an external software called Oracle Enterprise Manager that controls scheduled Oracle processes from the company.

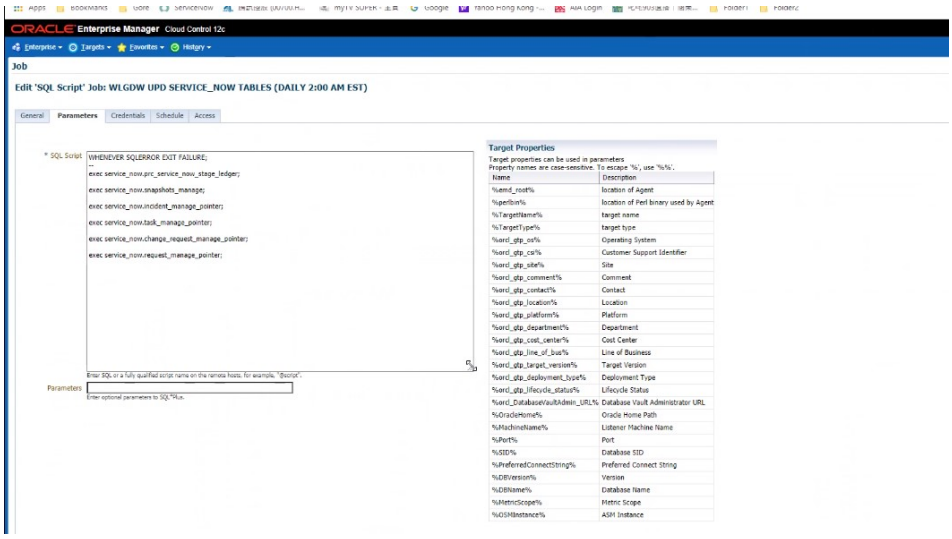


Fig. 3.11 Oracle Enterprise Manager script screenshot

3.2 Data warehouse implementation

To get to the current scheme of the ServiceNow data warehouse, I had to try other designs until I found the one that worked best. To create the tables I couldn't do that myself, it had to be done by the team in the company who is in charge of managing the data warehouse so I will not show the implementation process. In this section I will talk about the process to achieve the current design and the steps on this process.

The biggest issue to solve has been the performance, at the beginning I tried to build dashboards using the ledger tables but due the large amount of records in these tables it was impossible, the dashboards were giving errors all the time. Then I started using a view called "Counts By Snapshot" which was a combination of the "By Snapshot" views mentioned in previous sections that contained all ticket types in the same table. My idea was to add this table to the data module and then filter the snapshot flags there, but again it was too slow, this was due that every time that the data module was pulling data from the view, the server had to recreate the view and this was too much.

Finally after many tries I decided to create materialized views, because as I explained at the beginning of section 2.3 the difference between them and the views is that the results of a view query are not stored anywhere on disk, and

the view is recreated every time the query is executed. Materialized views are actual structures stored within the database and written to disk. So I decided which are the columns that I need to create and asked the data team to create “Combined Tickets Current”, “Combined Tickets Monthly Ind”, and “Combined Tickets Monthly Summary”. At this point the performance was better but not enough so we had to create indexes in the most relevant fields. Indexes in SQL are like keys stored in a structure that helps the server to find the rows associated with the key values quickly and efficiently.

	INDEX_OWNER	INDEX_NAME	COLUMNS	UNIQUENESS	STATUS	INDEX_TYPE	TEMPORARY	PARTITIONED
1	SERVICE_NOW	COMBINED_TICKETS_CURR_B1	ASSIGNMENT_GROUP	NONUNIQUE	VALID	BITMAP	N	NO
2	SERVICE_NOW	COMBINED_TICKETS_CURR_B2	ASSIGNED_TO	NONUNIQUE	VALID	BITMAP	N	NO
3	SERVICE_NOW	COMBINED_TICKETS_CURR_B3	SYS_CLASS_NAME	NONUNIQUE	VALID	BITMAP	N	NO
4	SERVICE_NOW	COMBINED_TICKETS_CURR_B4	COUNT	NONUNIQUE	VALID	BITMAP	N	NO
5	SERVICE_NOW	COMBINED_TICKETS_CURR_IND1	CLOSED_AT_YEAR	NONUNIQUE	VALID	NORMAL	N	NO
6	SERVICE_NOW	COMBINED_TICKETS_CURR_IND2	CLOSED_AT_MONTH	NONUNIQUE	VALID	NORMAL	N	NO

Fig. 3.12 Indexes on Combined Tickets Current table

Another problem that we had is that for a period of time ODI had recorded all data using a property of the fields called the label instead of using the the name, the result was an inconsistency on the records, the records were recorded in two different ways as shown in Fig. 3.13.

SYS_CLASS_NAME
Catalog Task
Change Request
Incident
change_request
incident
sc_task

Fig. 3.13 Records inconsistency

This was reflected on the dashboards creating confusing results like Fig. 3.14.

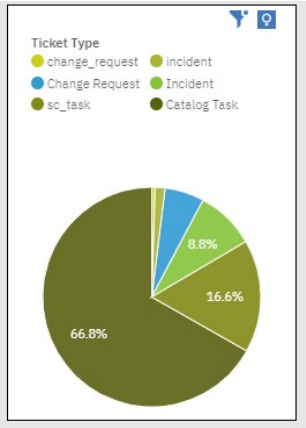


Fig. 3.14 Dashboard confusing result due data inconsistency

To solve that problem we had to do a massive SQL update to unify the records format in the ledger tables.

3.3 Creating our Data Module

I had to recreate the Data Module many times, each modification in the data warehouse was creating conflicts so I had to create a new Data Module from scratch. In this section I will show how I created the most relevant parts of the Data Module and I will explain the issues that I found doing that.

3.3.1 Creating the views

As previously mentioned in section 2.4.2 I created three new tables in the data module based on the table Combined Tickets Current. These tables only contain one ticket type and are used for analyzing the individual trends. Cognos Data Module provides the options seen in Fig. 3.X.

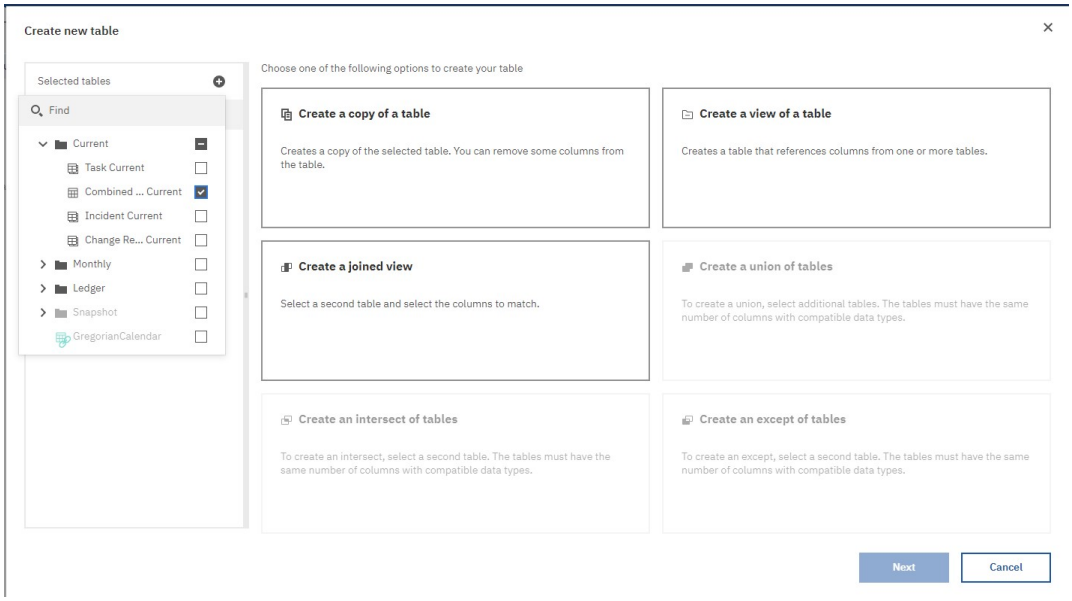


Fig. 3.15 Create a new table

I selected the option that creates a copy of the table selected and didn't remove any column. I repeated this process three times and after having these 3 exact copies of the Combined Tickets Current table I filtered each of them by the ticket type as seen in Fig. 3.16.

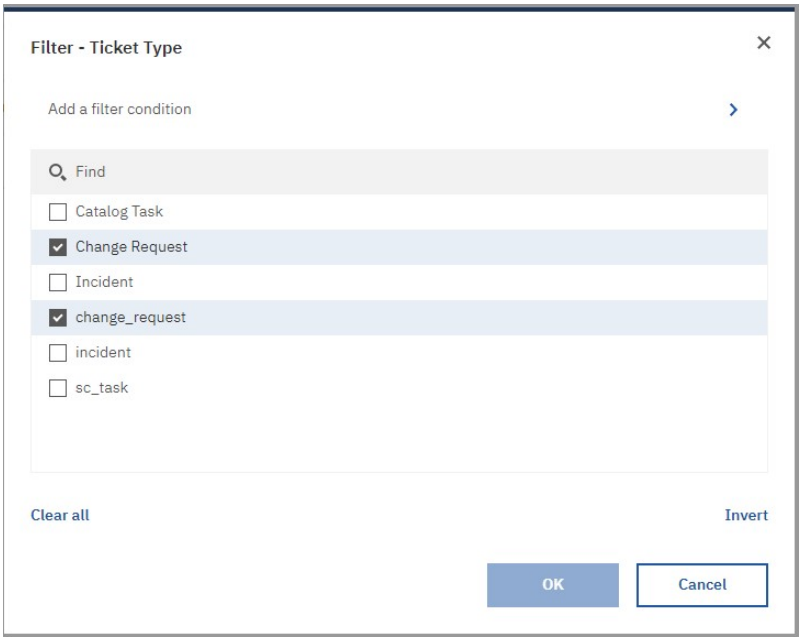


Fig. 3.16 Filter applied in the Change Request table

3.3.2 Shaping the data

Another possibility of the Data Module is to define the data format of the columns, to define the usage of the column, how to aggregate it and many more properties. This will be very useful in the next step so the analysis tools will understand the data and will display it better. I will show some property definitions that I defined below.

Fig. 3.17 is showing the properties for the Ticket Number column, where the "Usage" is "Identifier", this will be useful if we want to create relationships between tables. The Aggregate field defines what Cognos have to do in case of using this column in an analysing tool that aggregates data, the option "Count Distinct" will count all the rows that have different values.

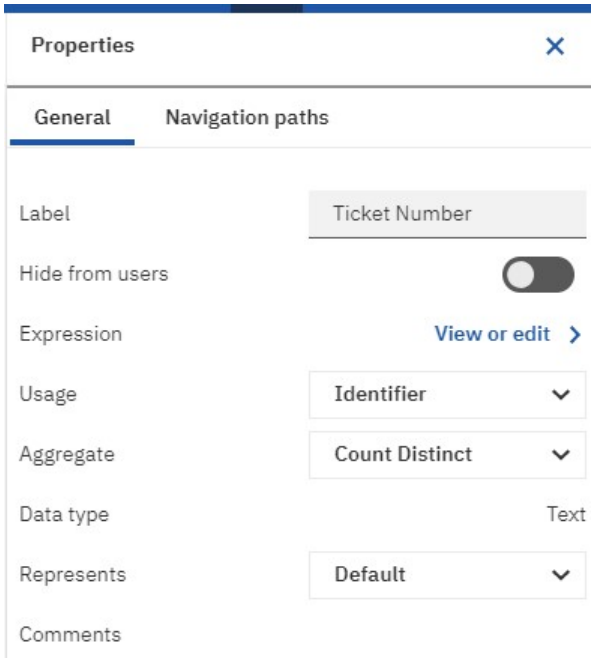


Fig. 3.17 Ticket Number properties

In Fig. 3.18 you can see the properties of the column Count, this column is always "1" in every row and it is used for counting rows in different ways. The field "Usage" is set up as "Measure" so it will be easier for Cognos to manage it, and the field "Aggregate" is set up as "Count", so Cognos will sum all the rows if has to aggregate the column in an analytic tool.

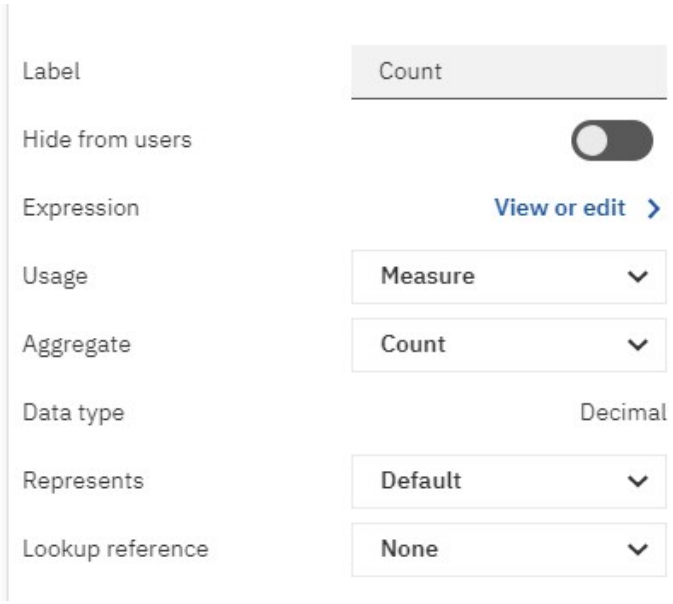


Fig. 3.18 Count properties

There is another important field that helps Cognos to interpret the data, the field “Represents” tells Cognos if the column represents time or a geographic region, this will be useful in the next step when creating map representation and other graphical tools. Fig. 3.19 shows the properties of the column “Opened At” that records the date and time when the ticket was opened. In this case I defined the field “Aggregate” to “None” because I don’t want Cognos to aggregate this column in any case.

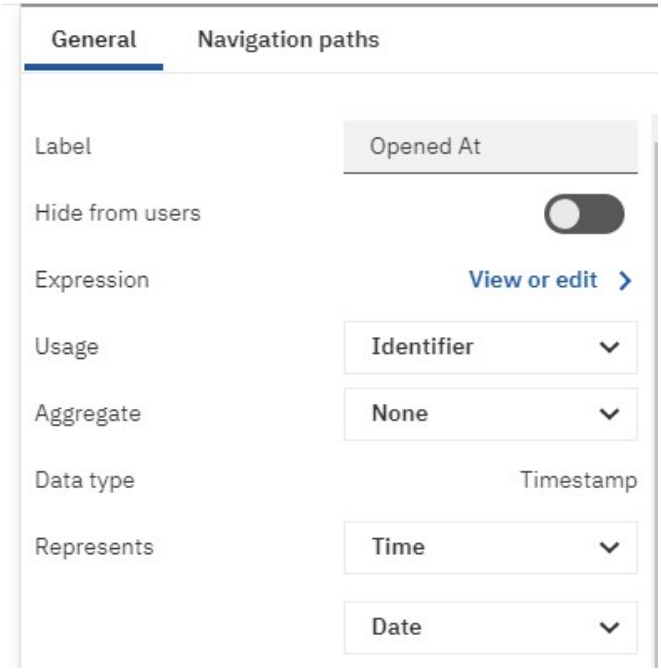


Fig. 3.19 Opened At properties

Data Module also brings the possibility of defining how the data will be displayed in tables or other graphical tools. In Fig. 3.20 you can see an example of the data format for the column “Opened At”.

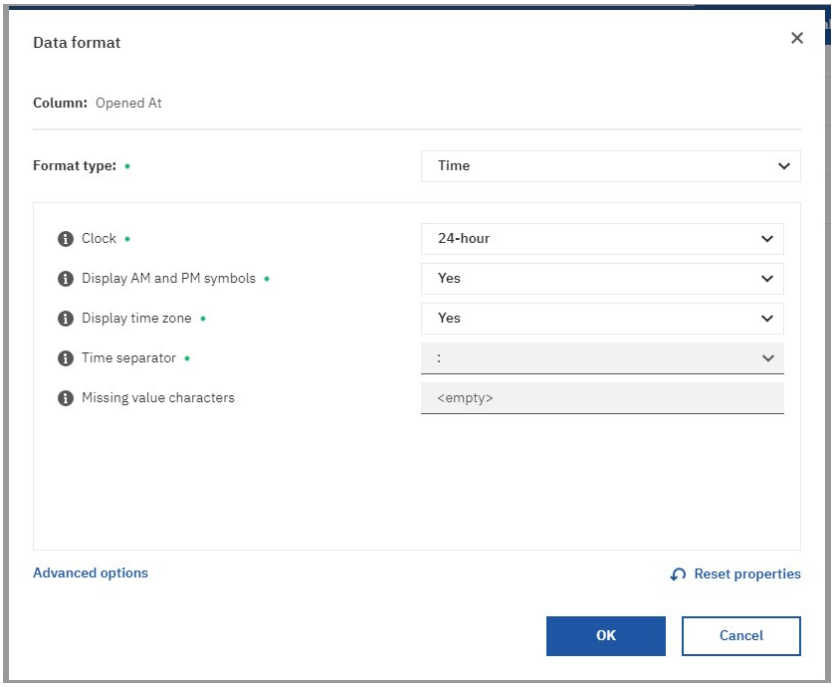


Fig. 3.20 Opened At data format options

3.3.3 Creating filters

The filters in the Data Module are defined using a coding language created by Cognos that is very similar to SQL.

Fig. 3.21 and 3.X show the interface to create the filters. On the left side, we have the columns from the table and some functions that we can drag it to the expression section, on the right side, where we will define the rules.

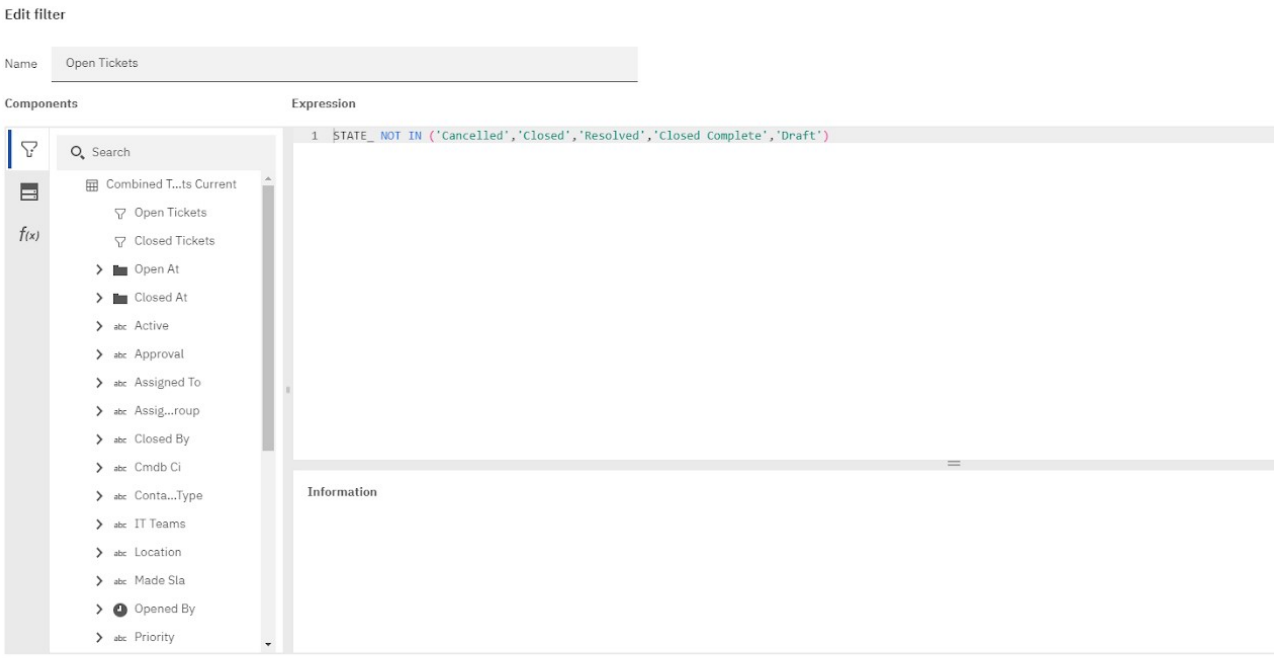


Fig. 3.21 Open Tickets filter definition

Fig. 3.22 Closed Tickets filter definition

In both cases we are using the column “State”, a ticket can have many different states but when analysing trends sometimes we don’t need that much detail, we just want to know if it is open or closed. The definition for Closed Tickets filter says: When column STATE_ is equal to one of the states inside the brackets, then the ticket is closed. And the Open Tickets filter is doing the opposite.

3.3.4 Creating the calculations

Again the calculations are defined using the coding language by Cognos, the same that we used to create the filters, and the interface to create the calculations is also very similar to the interface to create the filters.

To create different time ranges for the Opened At and Closed At attributes what I did is to use the original columns, as they have all the information about day, month, year, and even the hour, and extract the interesting part for each calculation.

For example I wanted to have a column that displays the month when a ticket was closed, to do so I created the code shown in Fig. 3.23.

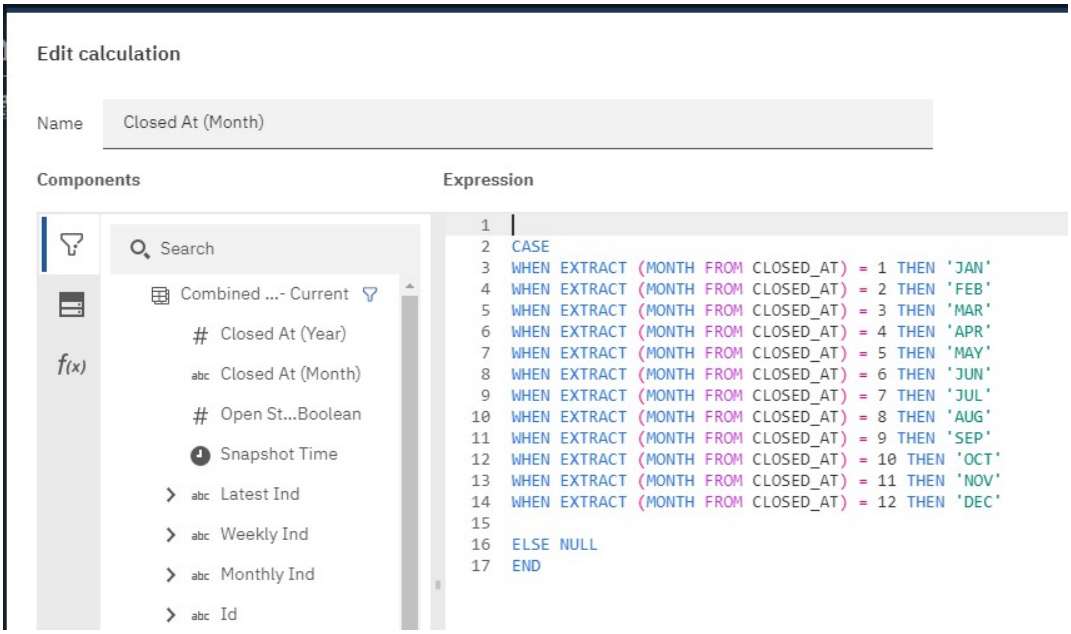


Fig. 3.23 Closed At Month definition

This code is creating a condition saying that when extracting the month from the column Closed At and it is equal to 1, then the column Closed At Month will be equal to ‘JAN’, when it is equal to 2, column Closed At Month will be ‘FEB’, and like this till December. If none of these conditions is fulfilled, then the value will be null.

Another interesting calculation was to have a column that shows the year and the month when the ticket was opened, it is very useful to order chronologically.

Fig. 3.24 Opened At (Year/Month) definition

The code for Opened At (Year/Month) is very simple, it just extracts the year and the month from the Opened At column and formats it.

The calculation Opened At Year is using the same method as the calculations above as seen in Fig. 3.25.

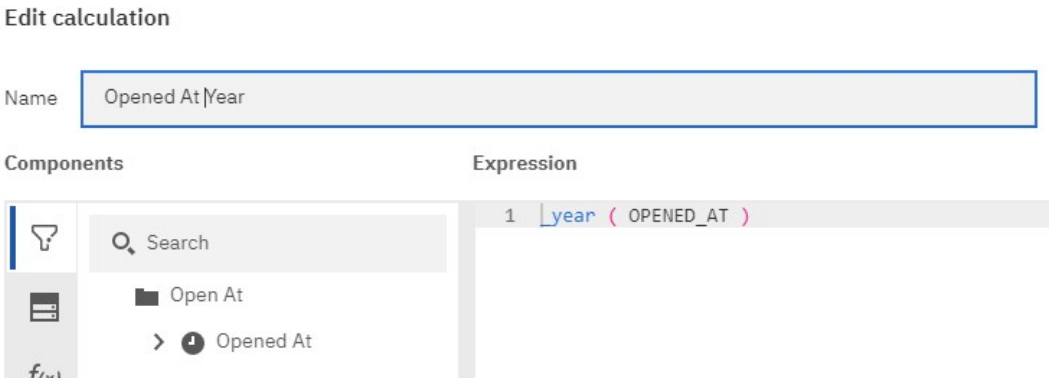


Fig. 3.25 Opened At Year definition

I created all calculations for the Opened At and Closed At columns on all tables in the data module.

3.3.5 Link to the Gregorian Calendar

To use the Gregorian Calendar I just had to add it to the data module as a new source, as shown in Fig. 3.26.

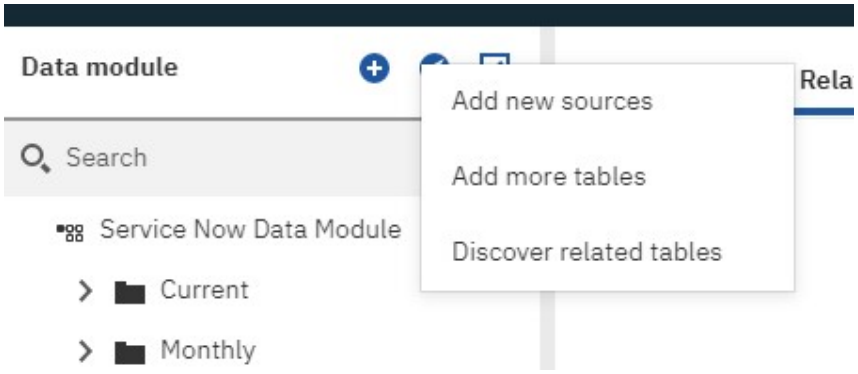


Fig. 3.26 Add to data module menu

After adding the calendar from the public folders from Cognos it already appears under the other tables as another table.

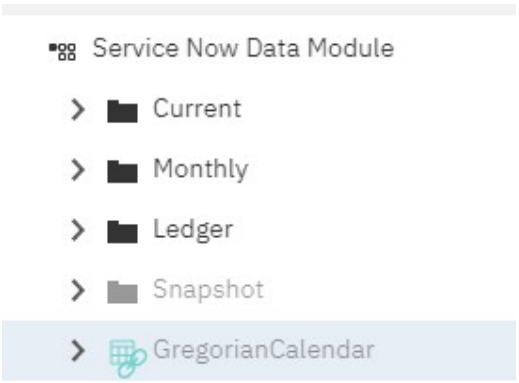


Fig. 3.27 Tables from ServiceNow Data Module

Then, to link the filters from the Gregorian Calendar to be used in a certain column is very easy, just have to open the properties from the column and select Gregorian Calendar in a field called Lookup Reference.

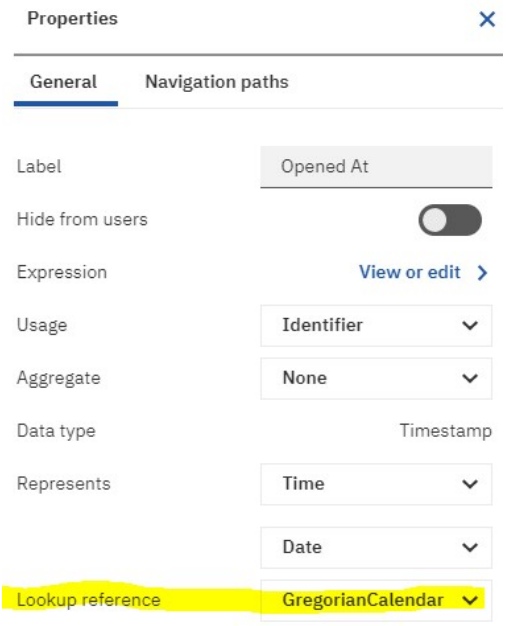


Fig. 3.28 Opened At properties

After this, the filters from the Gregorian Calendar can be used in the column Opened At.

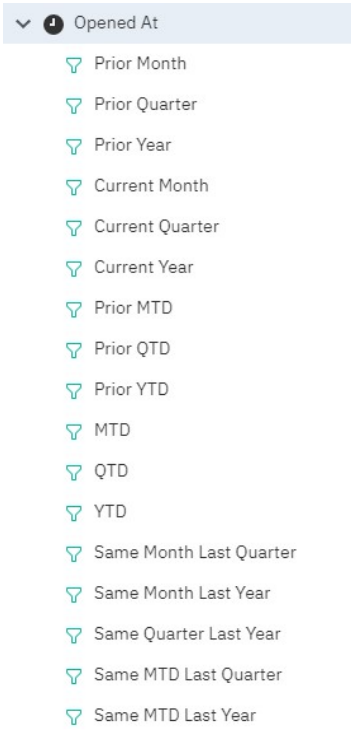


Fig. 3.29 Gregorian Calendar Filters on Opened At column

3.3.6 Creating a data group

To create the data group you have to choose the column on which the data group will be based, in our case it is the column “Assignment Group”, right-click and select the option “Create data group…

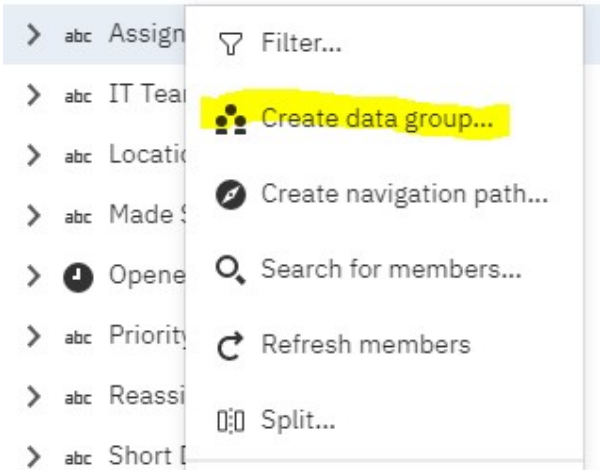


Fig. 3.30 Right-click menu on Assignment Group

Then Cognos will open a very user-friendly interface to create the data group as seen in Fig. 3.31.

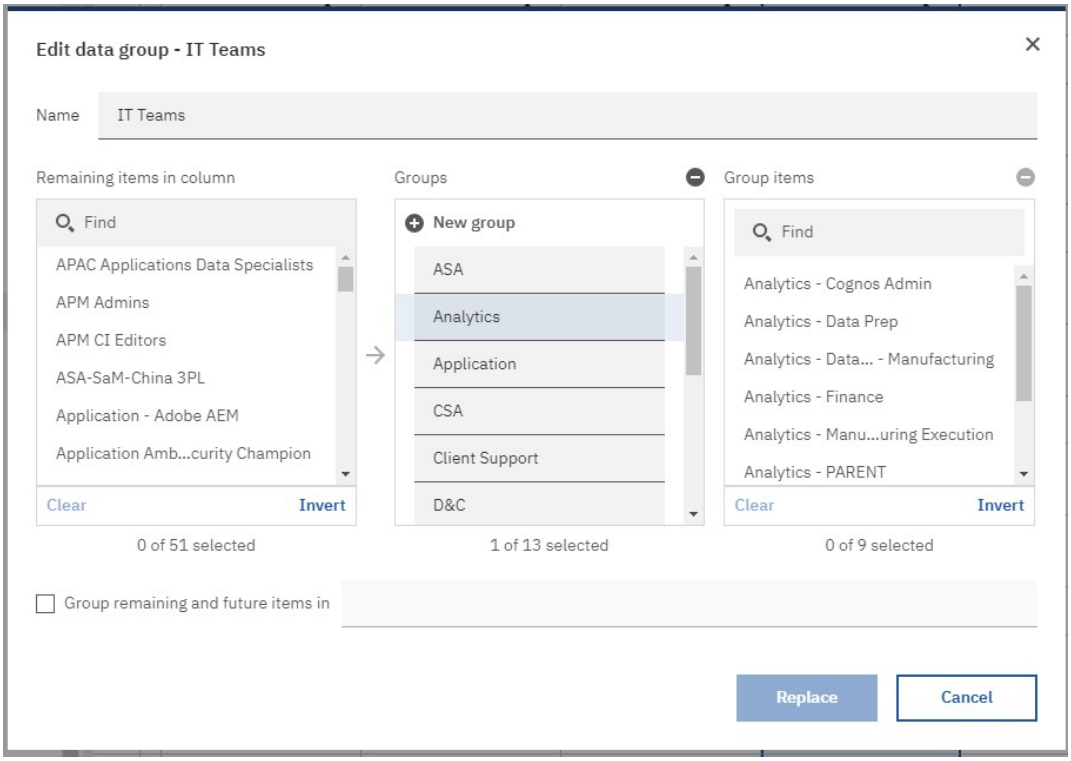


Fig. 3.31 Right-click menu on Assignment Group

The interface is divided in three parts, in the middle there is a space to create the groups, on the left side you can see all the items that don't have a group, and on the right side the items inside the group selected. To add an item to a group it just has to be selected and click the arrow.

3.4 Creating the dashboards

In this section I will explain how I created the key features of the dashboards, what I used, and which issues I found that I didn't expect. Creating the visualizations and filters is quite easy with the Cognos tools, the difficulty is to find the way to combine the columns from the tables in the data module to display information that an user can understand easily. I will divide this section in subsections by kinds of objects and processes.

3.4.1 Filters

There are 2 types of filters, filters that apply to all tabs, and filters that apply to just one tab.

To create a filter list you just have to drag one column from the tables located on the left side of the screen to the bar in the top. As shown in Fig. 3.32 depending on which side you drop it, it will apply to all tabs or just to this tab.

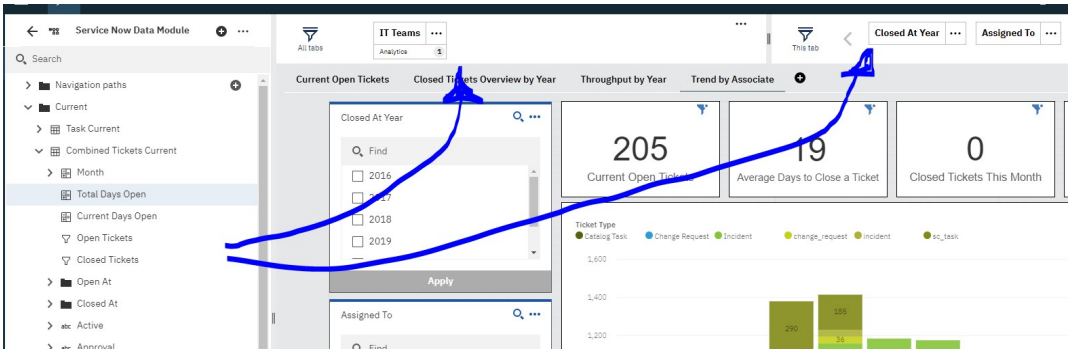


Fig. 3.32 Creating filters

Then these filters can be added to the canvas so it is more user-friendly, to do so, there is an option clicking the three dots in the right side of the filter name called "Add to canvas".

Here you can see the reason why I created the calculation Closed At Year, at the beginning I wanted to filter the data by years but I couldn't with the Closed At column coming from the data warehouse because it was showing all dates

with day and month, then I figured out that I need a column that just displays the year.

The objects in the tab also work as filters, in the way that if for example in the stacked bar graph from the Trend by Associate tab you click on a bar (that represents a month) all other objects will be filtered by that month. There is a way to create separated groups so one object will only filter the objects in the same group. This can be done clicking the button shown in Fig. 3.33.

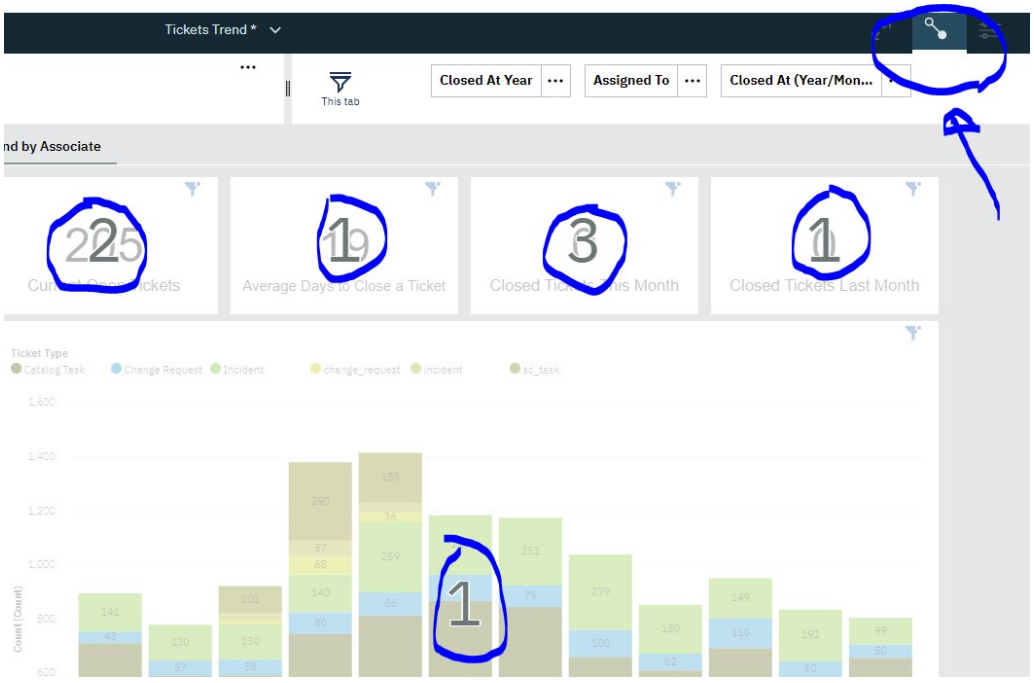


Fig. 3.33 Filter groups

Fig. 3.33 shows different groups of objects. If you click one bar from the graph (group 1), only the objects in group 1 will be filtered.

3.4.2 Current tickets table

I wanted to create a table that shows basic information of the current open tickets to analyze the state of the ServiceNow and the teams. Talking with people from the company we decided that it would be interesting to see how many times this ticket was reassigned from one group to another, if it made the SLA (Service Level Agreement), and how many days have passed since it was opened.

I didn't have a column showing the current days open so I had to create a calculation, it was too specific to create it in the Data Module as I would only use it on this dashboard so I created the calculation in the dashboard as shown in Fig. 3.34.

Fig. 3.34 Current Days Open calculation

A very simple calculation counting the days between the date when the ticket was opened and the current date.

Then I opened the visualizations menu and selected a table as shown in Fig. 3.35.

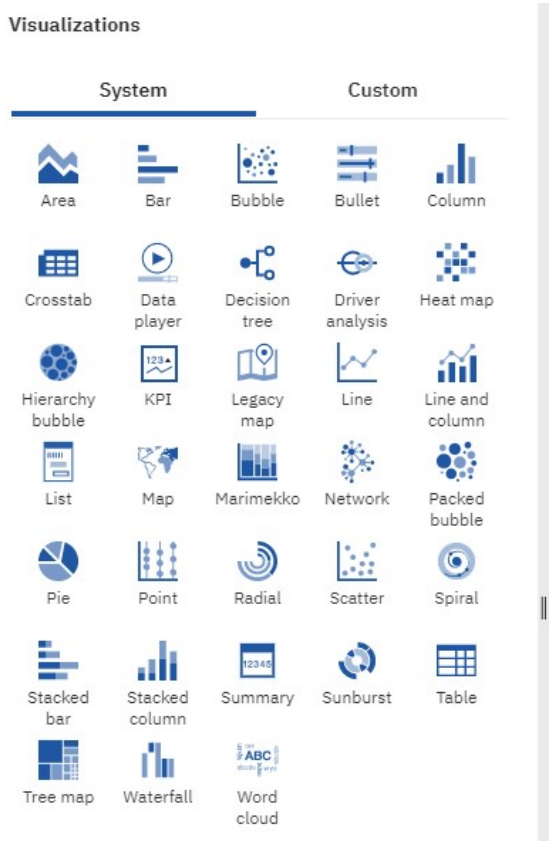


Fig. 3.35 Visualizations menu

After that, an interface that has three parts comes up, as shown on Fig. 3.36, on the first space I dragged the columns that I want to have on the table, the following section lets you create color rules on the table, in this case I wanted to paint in red the tickets that have been open for more than a year so I dragged the column Current Days Open. The third section is for filters that apply only to this visualization, I dragged the previously created Open Tickets filter so the table will only show data from open tickets.

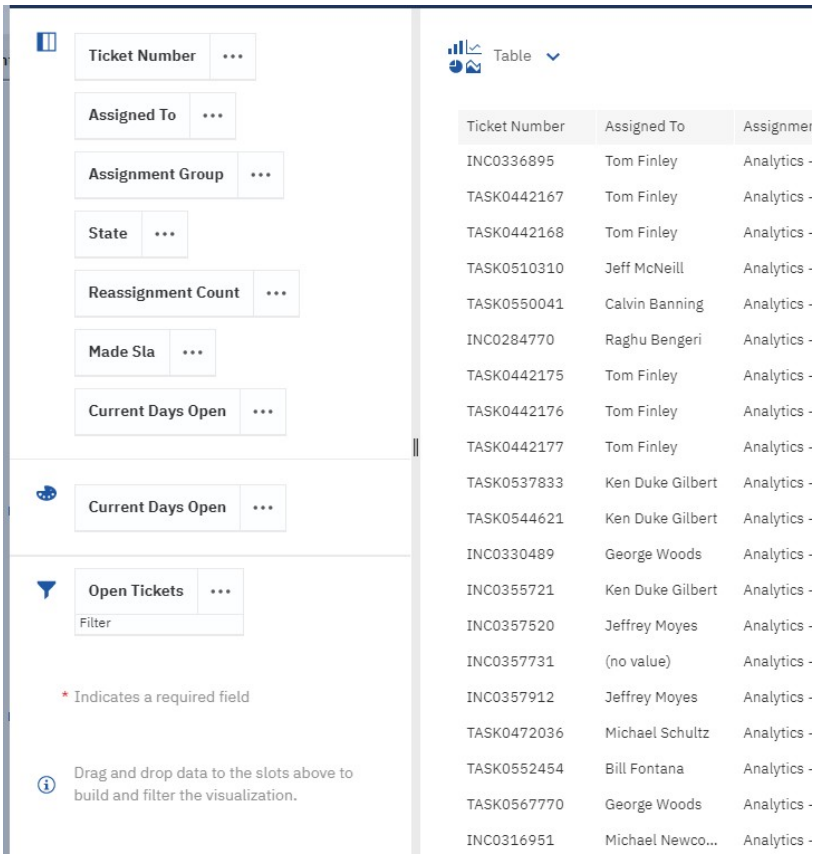


Fig. 3.36 Interface to create a table

Then I had to create a custom color palette and define the rules to paint the tickets in red as seen in Fig. 3.37 and 3.38.

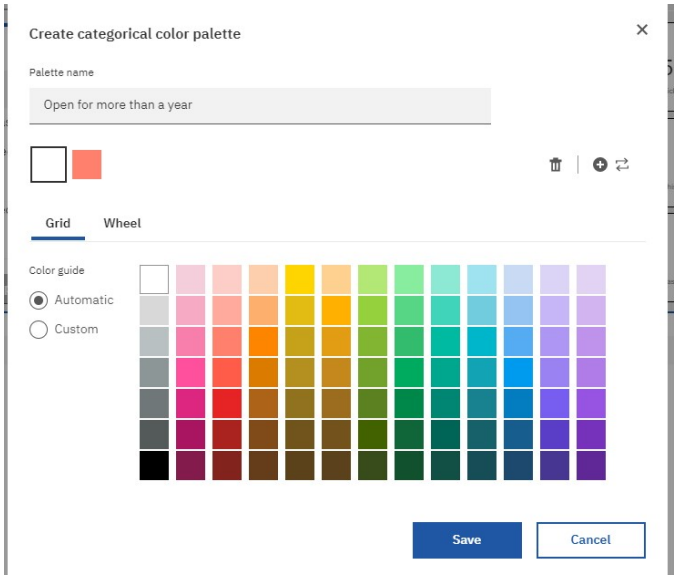


Fig. 3.37 Interface to create a custom color palette

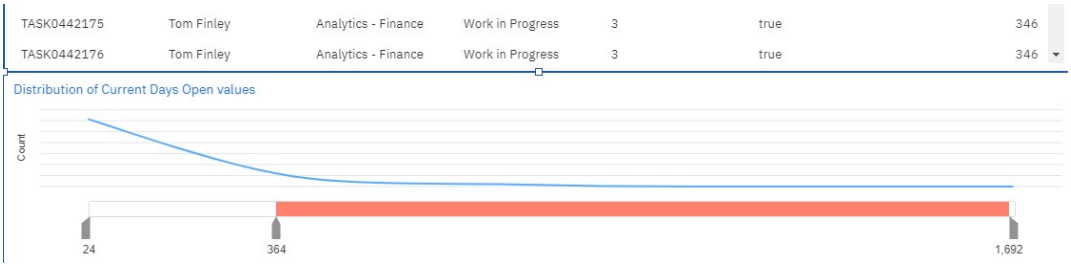


Fig. 3.38 Interface to create a color rule

3.4.3 KPIs

To create the KPIs it is a very similar process than to create other visualizations as the table mentioned in the previous section.

I wanted to create a KPI showing the average days that a ticket remained opened, as shown in Fig. 3.39.

Fig. 3.39 Average Days Open KPI

In the data warehouse we have individual records for each ticket, but we do not have general data so to have a measure that calculates the average time that a ticket remains open I had to create the calculation Total Days Open, counting the days between the Opened At column and the Closed At column, then drag this calculation to a KPI and select the summarize option Average.

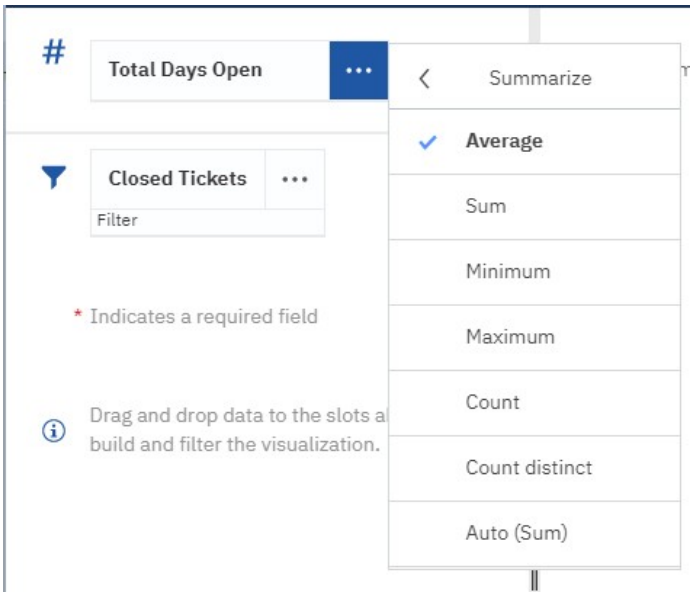


Fig. 3.40 KPI creation interface

Then I also dragged the filter closed tickets so the KPI will only show data from tickets that are already closed.

I created a KPI that shows the number of tickets closed last month, to do so, I used the Prior Month filter from the Gregorian Calendar applied to the Closed At column, explained in chapter 2 and the column count as seen in Fig 3.41.

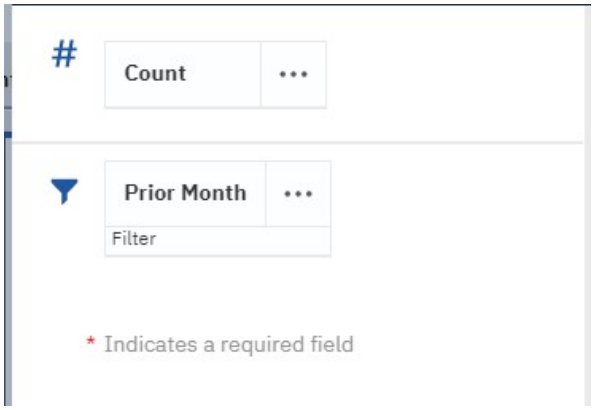


Fig. 3.41 Interface to create the Closed Tickets Last Month KPI

As I wanted to know the number of tickets closed, I had to tell Cognos to Count the number of positions in the column Count, as shown in Fig. 3.42.

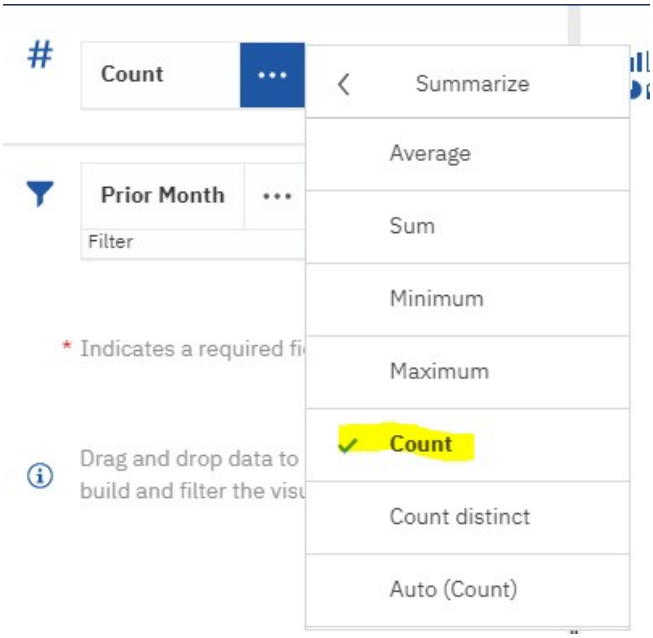


Fig. 3.42 Summarize options

3.4.4 Throughput by Year graph

I wanted to create a visualization that compares the number of tickets opened and the number of tickets closed each month. To do so, I created a line and column graph as shown in Fig. 3.43.

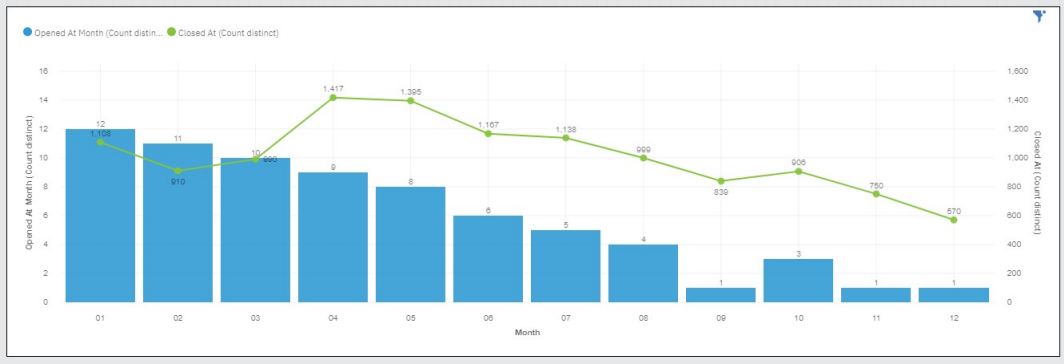


Fig. 3.43 Throughput by Year graph

The problem that I had is that, as shown in Fig. 3.44, the interface to create the graph only let me select one data column to represent the X axis.

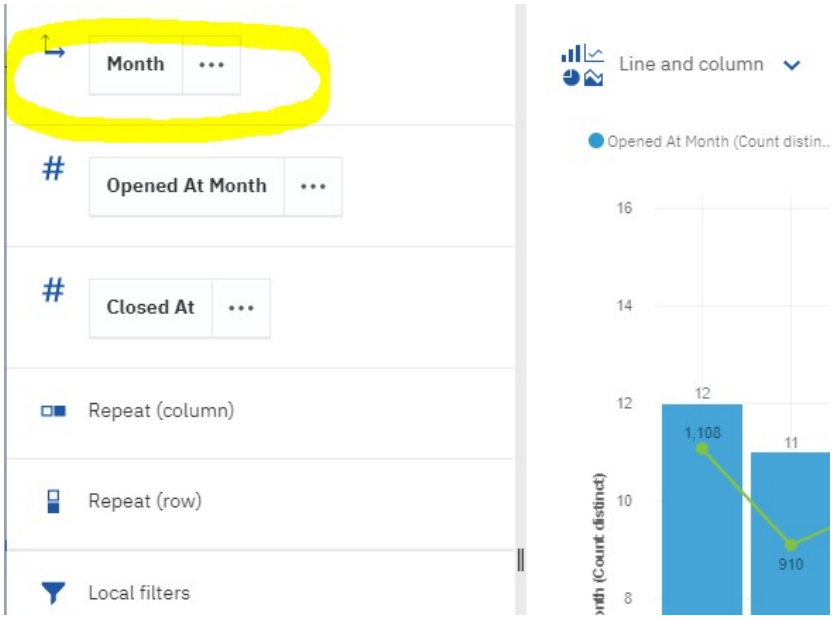


Fig. 3.44 Throughput by Year graph

I had to count the number of tickets opened and closed at the same month, but these data for each record is separated in two different columns: Opened At and Closed At, so I had to create a new calculation called Month that joins both values.

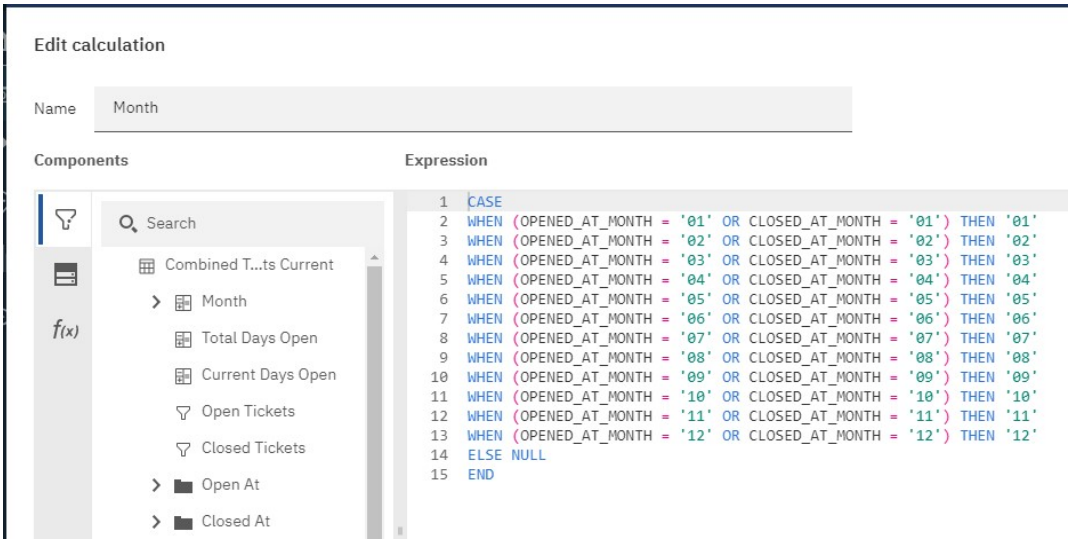


Fig. 3.45 Logic on Month calculation

This logic is saying that when one of the columns Opened At Month or Closed At Month is equal to the value “01”, then the value displayed will be “01” (that will represent January), then it iterates the same logic for all month till December.

Chapter 4. Application example

In this chapter we will see how a ticket in the ServiceNow travels through the system to finally appear on the dashboards. We will take as an example a ticket assigned to me, it is a Change Request to create a new report for the customer service in Germany, and the ticket number is “CHG0027217”.

In Fig. 4.1 We can see the ticket in ServiceNow located in the tab “My Work”.

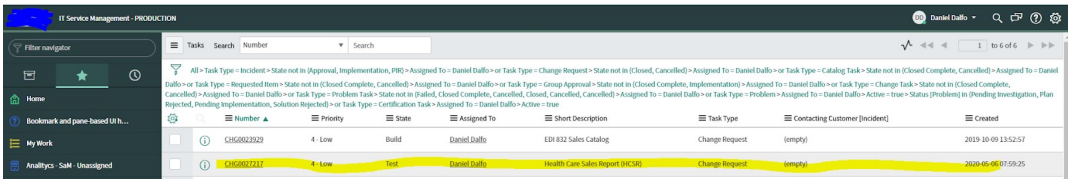
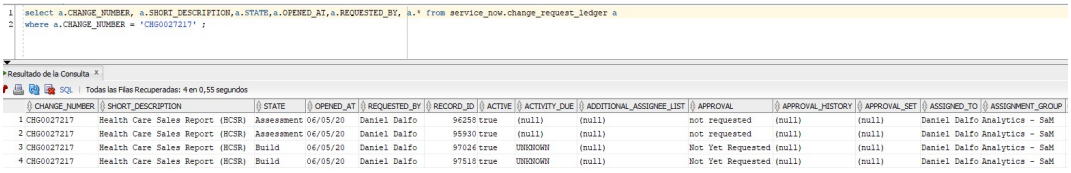


Fig. 4.1 My Work tab on ServiceNow

As we have seen in previous chapters, the data from this ticket is recorded in the ServiceNow database, then extracted by the ODI, loaded in the stage table for Change Requests in the data warehouse, and from the stage table loaded in the Change Request ledger table. Fig. 4.2 shows all the records for this ticket number on the ledger table.



```
select a.CHANGE_NUMBER, a.SHORT_DESCRIPTION,a.STATE,a.OPENED_AT,a.REQUESTED_BY, a.* from service_now.change_request_ledger a
where a.CHANGE_NUMBER = 'CHG0027217' ;
```

Resultado de la Consulta

SQL | Todas las Filas Recuperadas: 4 en 0,55 segundos

	CHANGE_NUMBER	SHORT_DESCRIPTION	STATE	OPENED_AT	REQUESTED_BY	RECORD_ID	ACTIVE	ACTIVITY_DUE	ADDITIONAL_ASSIGNEE_LIST	APPROVAL	APPROVAL_HISTORY	APPROVAL_SET	ASSIGNED_TO	ASSIGNMENT_GROUP
1	CHG0027217	Health Care Sales Report (HCSR)	Assessment	06/05/20	Daniel Dalfo	96258	true	(null)	(null)	not requested	(null)	(null)	Daniel Dalfo	Analytics - SaM
2	CHG0027217	Health Care Sales Report (HCSR)	Assessment	06/05/20	Daniel Dalfo	95930	true	(null)	(null)	not requested	(null)	(null)	Daniel Dalfo	Analytics - SaM
3	CHG0027217	Health Care Sales Report (HCSR)	Build	06/05/20	Daniel Dalfo	97026	true	UNKNOWN	(null)	Not Yet Requested	(null)	(null)	Daniel Dalfo	Analytics - SaM
4	CHG0027217	Health Care Sales Report (HCSR)	Build	06/05/20	Daniel Dalfo	97518	true	UNKNOWN	(null)	Not Yet Requested	(null)	(null)	Daniel Dalfo	Analytics - SaM

Fig. 4.2 Change Request ledger table

We can see that the ledger table contains more than one record for the ticket, and watching the column State we will see that there are records for the different phases of the ticket. Some of the records were taken when the state of the ticket was “Assessment” and some when the state was “Build”.

From the ledger table then will be mixed with tickets of other types and loaded into the materialized views. Fig. 4.3 shows the records for this ticket in the Combined Tickets Current materialized view used on the dashboards.

```
4  select a.TICKET_NUMBER,a.STATE,a.OPENED_AT, a.*
5  from SERVICE_NOW.COMBINED_TICKETS_CURRENT a
6  where a.TICKET_NUMBER = 'CHG0027217' ;
```

Resultado de la Consulta

SQL | Todas las Filas Recuperadas: 1 en 0,591 segundos

	TICKET_NUMBER	STATE	OPENED_AT	SNAPSHOT_TIME	TICKET_NUMBER_1	SHORT_DESCRIPTION	COUNT	ACTIVE	APPROVAL	ASSIGNED_TO	ASSIGNMENT_GROUP
1	CHG0027217	Build	06/05/20	01/08/20	CHG0027217	Health Care Sales Report (HCSR)	1	true	Not Yet Requested	Daniel Dalfo	Analytics - SaM

Fig. 4.3 Combined Tickets Current table

This time we only have one record for the ticket, this is because the Combined Tickets Current table only has records for the current state of the ticket.

To see the current state tickets we will use the Tickets Trend dashboard and one good way to see data by individual tickets is using the first tab called Current Open Tickets. To find the ticket we will use the filters on the tab. Filtering by Ticket Type = “Change Request”, Assignment Group = “Analytics SaM” (that is actually my team), and Assigned To = “Daniel Dalfó” in Fig 4.4 we can see the table with the ticket number “CHG0027217”.

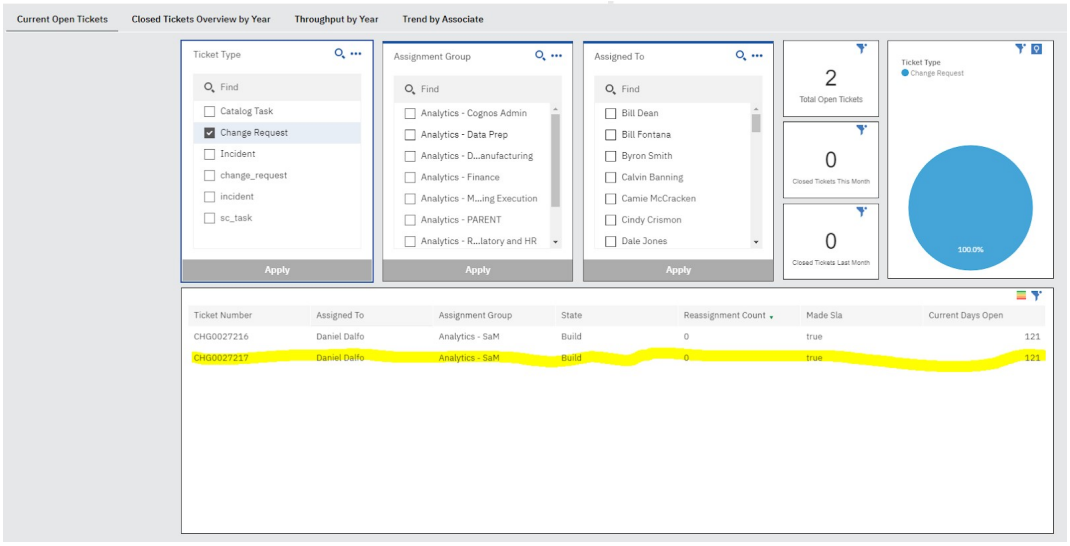


Fig. 4.4 Current Open Tickets filtered

Chapter 5. Conclusions

It is a fact that data science and business intelligence are becoming more and more important to companies every day. These kinds of systems help prevent wasting lots of money and resources by identifying dysfunctions in processes, needs, and new business opportunities.

Working on this project helped me to find a passion into the data science and business intelligence world, getting deep and investigating these subjects increased so much my interest for them.
The fact of working in a real company, doing things related to data science and business intelligence on a daily basis and seeing how they really work and how people work, has helped me understand in a much more conscious way many concepts that theoretically would be much more diffuse.

Some conclusions that I have drawn is that in the process of a data system there are many key factors such as performance, the way the data is grouped or the format we give it.

Handling such large volumes of data entails great precision in the logic that will guide this data on the way to being analyzed and small changes are reflected in great magnitude, so you have to be very careful in the process because you may be showing erroneous data.

The ServiceNow Data Extraction System will be used by real users in the company and it will continue to be supported and developed, this generates great satisfaction since the general objectives of the project have been met.